Donglei Du
Lu Han
Dachuan Xu (Eds.)

LNCS 15435

# Combinatorial Optimization and Applications

17th International Conference, COCOA 2024
Beijing, China, December 6–8, 2024
Proceedings, Part II

2 Part II

Springer

Donglei Du · Lu Han · Dachuan Xu
Editors

# Combinatorial Optimization and Applications

17th International Conference, COCOA 2024
Beijing, China, December 6–8, 2024
Proceedings, Part II

Springer

*Editors*
Donglei Du
University of New Brunswick
New Brunswick, NB, Canada

Lu Han
Beijing University of Posts
and Telecommunications
Beijing, China

Dachuan Xu
Beijing University of Technology
Beijing, China

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-981-96-4447-6 ISBN 978-981-96-4448-3 (eBook)
https://doi.org/10.1007/978-981-96-4448-3

# Preface

The 17th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2024) took place in Beijing, China, during December 6–8, 2024. COCOA 2024 provided an excellent venue for researchers in the area of combinatorial optimization and its applications, including algorithm design, theoretical and experimental analysis, and applied research of general algorithmic interest. The Program Committee received a total of 124 submissions, among which 53 were accepted for presentation at the conference. Each contributed paper was subject to a rigorous peer review process, receiving three double-blind reviews from reviewers selected from the Program Committee. We would like to express our sincere appreciation to everyone who made COCOA 2024 a success by volunteering their time and effort: the authors, the Program Committee members, and the reviewers. We thank Springer for accepting the proceedings of COCOA 2024 for publication in the Lecture Notes in Computer Science (LNCS) series. Our special thanks also extend to the other chairs and the conference Organizing Committee members for their excellent work.

Donglei Du
Lu Han
Dachuan Xu

# Organization

## General Chair

| | |
|---|---|
| Du, Ding-Zhu | University of Texas at Dallas, USA |

## Program Committee Co-chairs

| | |
|---|---|
| Du, Donglei | University of New Brunswick, Canada |
| Han, Lu | Beijing University of Posts and Telecommunications, China |
| Xu, Dachuan | Beijing University of Technology, China |

## Web Co-chairs

| | |
|---|---|
| Sai Ji | Hebei University of Technology, China |
| Lili Mei | Hangzhou Dianzi University, China |

## Publication Co-chairs

| | |
|---|---|
| Chenchen Fu | Southeast University, China |
| Ruiqi Yang | Beijing University of Technology, China |

## Finance Chair

| | |
|---|---|
| Yicheng Xu | Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China |

## Registration Chair

| | |
|---|---|
| Jian Sun | Nanjing Normal University, China |

## Local Chairs

| | |
|---|---|
| Zhongzheng Tang | Beijing University of Posts and Telecommunications, China |
| Chenchen Wu | Tianjin University of Technology, China |

## Program Committee Members

| | |
|---|---|
| Bein, Wolfgang | University of Nevada, USA |
| Calinescu, Gruia | Illinois Institute of Technology, USA |
| Chau, Vincent | Southeast University, China |
| Chen, Xujin | Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China |
| Chen, Yong | Hangzhou Dianzi University, China |
| Cheng, Yukun | Jiangnan University, China |
| DasGupta, Bhaskar | University of Illinois Chicago, USA |
| De Bonis, Annalisa | Università degli Studi di Salerno, Italy |
| Diao, Zhuo | Central University of Finance and Economics, China |
| Fan, Neng | University of Arizona, USA |
| Feng, Qilong | Central South University, China |
| Guo, Longkun | Fuzhou University, China |
| Ji, Sai | Hebei University of Technology, China |
| Jiang, Shaofeng | Peking University, China |
| Khachay, Michael | Krasovsky Institute of Mathematics and Mechanics, Russia |
| Lee, Joong-Lyul | University of North Carolina at Pembroke, USA |
| Li, Xianyue | Lanzhou University, China |
| Li, Minming | City University of Hong Kong, China |
| Li, Jianping | Yunnan University, China |
| Lin, Guohui | University of Alberta, Canada |
| Liu, Bin | Ocean University of China, China |
| Mei, Lili | Hangzhou Dianzi University, China |
| Nguyen, Viet Hung | Clermont Auvergne University, France |
| Ogihara, Mitsunori | University of Miami, USA |
| Satpute, Meghana | University of Texas at Dallas, USA |
| Tang, Zhongzheng | Beijing University of Posts and Telecommunications, China |
| Tong, Weitian | Eastern Michigan University, USA |
| Wu, Chenchen | Tianjin University of Technology, China |

| | |
|---|---|
| Xu, Yicheng | Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China |
| Yang, Ruiqi | Beijing University of Technology, China |
| Zhang, Ruilong | Technical University of Munich, Germany |
| Zhang, Peng | Shandong University, China |
| Zhang, Zhao | Zhejiang Normal University, China |
| Zhang, Xiaoyan | Nanjing Normal University, China |
| Zhang, Yong | Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China |
| Ziegler, Martin | Korea Advanced Institute of Science and Technology, South Korea |
| Zissimopoulos, Vassilis | National and Kapodistrian University of Athens, Greece |

## Additional Reviewers

Chen, Lin
Chen, Xianrun
Dai, Sijia
Duan, Chengcheng
Filippos, Mavropoulos
Guo, Xinru
Ju, Jiachen
Khachai, Daniil
Lazaropoulos, Nikos
Li, Mengzhen
Lian, Yuefang
Liang, Wei
Lou, Jianing
Luo, Junjie
Polevoy, Gleb
Popa, Alexandru
Sigalas, Ioannis
Wang, Guihao
Wang, Fengjuan
Wang, Changjun
Wang, Chenhao
Xia, Xinlan
Xiao, Hao
Ye, Qingjie
Yu, Wei
Zhang, Yubai
Zhang, An
Zhang, Yubo
Zhao, Lei
Zoros, Dimitris

# Contents – Part II

# Contents – Part I

# Maximizing One-Way Trading Revenue in Photovoltaic Energy Generation

Xinru Guo[1,2], Yu Huang[3,4], Xinxin Han[5](✉), Yicheng Xu[1,2], Keliang Duan[6], and Qiancheng Xu[6]

[1] Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, People's Republic of China
{xr.guo,zhangyong,yc.xu}@siat.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, People's Republic of China
[3] Shenzhen University of Advanced Technology, Shenzhen, People's Republic of China
y.huang5@siat.ac.cn
[4] College of Mathematics and Information Science, Hebei University, Baoding, People's Republic of China
[5] School of Artificial Intelligence, Shenzhen Polytechnic University, Shenzhen, People's Republic of China
hanxin@szpu.edu.cn
[6] Beijing JH Eco-Energy Technology Co., LTD., Beijing 100160, People's Republic of China
{duankeliang,xuqiancheng}@jasolar.com

**Abstract.** Solar energy, particularly through photovoltaic (PV) power generation, plays a crucial role in transitioning to renewable energy sources, offering an inexhaustible and clean energy solution that significantly reduces carbon dioxide emissions and supports global carbon neutrality goals. Despite the rapid expansion of PV capacity worldwide, optimizing economic returns remains challenging due to market volatility and storage inefficiencies. During PV generation, produced energy is stored in batteries, waiting for favorable sale prices as the market fluctuates. In our model, storage losses for the batteries are represented by a constant, denoted as $C_{loss}$, which means that the remaining energy in the next period is reduced to $(1-C_{loss})$ times that of the previous period. This paper proposes a novel online algorithm designed to maximize revenue from selling PV power generated in each period. The algorithm dynamically adjusts selling strategies to effectively balance market price fluctuations and storage losses, achieving a competitive ratio of $O(\log h)$, where $h$ is the highest unit price. By strategically reserving $\frac{1}{\log h+1}$ of energy for each expected electricity unit price, the algorithm ensures greater profits at higher prices. Furthermore, the paper establishes that the lower bound of the trading problem is $\Omega(\log h)$, demonstrating that the algorithm is tight and performs optimally within the modeled constraints.

**Keywords:** One-way trading · Online algorithm · Competitive analysis · Photovoltaic

D. Du et al. (Eds.): COCOA 2024, LNCS 15435, pp. 1–13, 2025.
https://doi.org/10.1007/978-981-96-4448-3_1

## 1 Introduction

Solar energy, as an inexhaustible and clean energy source, offers hope for solving the energy crisis and significantly reduces environmental pollution, contributing to the goal of achieving carbon neutrality as soon as possible. Several studies have explored ways to improve the profitability of PV power generation and efficient energy delivery mechanisms [11]. The economic benefits of PV power generation vary among different user groups. Research by Grietus Mulder et al. indicates that household users gain more benefits from storing PV-generated electricity than from selling it and then repurchasing it [14]. Xin et al. designed a new control strategy to increase PV sales revenue, ensuring more efficient power output [18]. Furthermore, Ru et al. assessed the economic value of battery storage systems compared to purchasing power from the grid by studying the size of the batteries [15]. Therefore, this paper aims to design a mechanism that maximizes PV energy selling profit.

However, there is still a lack of research on the design of online algorithms for PV power sales. For such online pricing issues, Bansal et al. developed an online pricing algorithm for impatient bidders to maximize revenue [3]. Briest et al. demonstrated the non-approximability of polynomial-time algorithms for the unit demand minimum purchase pricing problem and extended it to a probabilistic consumer model [6]. Krysta et al. investigated the non-parametric unit demand pricing problem based on consumer profiles, proving the approximation difficulty of the minimum purchase model and exploring the maximum purchase and rank purchase models [7]. Chen et al. studied the envy-free pricing problem for revenue maximization in multi-item markets, proposing a polynomial-time algorithm and proving its feasibility under certain conditions [8]. Balcan et al. designed an online algorithm with an $O(k)$ approximation for maximizing seller revenue in the case of unlimited supply [2]. Bahram Alinia et al. designed an online algorithm for scheduling electric vehicles in adaptive charging networks to maximize social welfare under limited resources, proving that the online algorithm has a competitive ratio of 2 in fractional mode [1]. Xie et al. proposed that pre-sales could double the final revenue in their research on sales issues [17]. Avrim et al. designed an online algorithm for market clearing, achieving a competitive ratio of $\ln(p_{\max} - p_{\min}) + 1$ when bids are within the range of $[p_{\min}, p_{\max}]$ [5]. Kris Johnson Ferreira et al. developed dynamic pricing algorithms to maximize revenue in price-based network revenue management [10]. Zhang et al. presented new scenarios for online time series search and one-way trading problems, designing optimal deterministic online algorithms for these issues [20]. Avrim Blum et al. explored revenue maximization in online auctions, proposing a new online learning heuristic auction method that achieves a constant competitive ratio compared to the optimal offline fixed price revenue in digital goods auctions [4]. Ha et al. proposed an optimal intraday trading algorithm that optimally segments large market orders into a series of consecutive market orders to minimize overall trading costs, including liquidity and proportional trading costs [12]. Tan et al., in their pursuit of solving the optimal solution, designed an algorithm that determines whether to retain each element and also provides the quantity of retained elements [16]. Chin et al., in their study of a one-way trading problem, where a seller has L units of a product to sell to a series of buyers, designed an algorithm with a competitive ratio of $O(logr^* \cdot (log^{(2)}r^*) \cdot ... \cdot (log^{(h-1)}r^*) \cdot (log^{(h)}r^*)^{1+\epsilon})$ [9]. Han et al. proposed an approximation algorithm with a competition ratio of 5.9672 to study the PCkST problem [13]. Yang et al. proposed a threshold based BP flow maximization algorithm when studying flow maximization problem [19]. Zhang et al., in their research on minimizing bin packing costs, proposed an online bin packing strategy with a competitive ratio of 5.155 [21]. Zhang et al. maximized seller revenue in the sale of goods by setting the price and quantity for each item, achieving an $O(\log h)$ level result [22]. These findings inspire us to consider the application of online algorithms in PV power sales.

**Problem Description.** Energy generated by PV systems is sold to the grid for revenue. However, the unit price of electricity fluctuates, and favorable prices are expected. At the same time, energy stored in batteries incurs storage losses, represented by a constant $C_{\text{loss}}$ in this model. This means that the remaining energy in the next period is reduced to $(1 - C_{\text{loss}})$ times that of the previous period. Consequently, waiting too long for potentially higher sale prices may cost great storage losses. Therefore, decisions on the appropriate sale time must consider the balance between sale prices and storage losses.

PV generation occurs continuously online. However, in this paper, it is divided into discrete periods $i = 1, 2, 3, \dots$ for batch processing. It is assumed that at the beginning of each period $i$, both the PV output $P_i$ and the unit electricity price $R_i$ can be revealed. These inform decisions regarding the amount of electricity to sell immediately and the amount to reserve for subsequent periods.

The goal of this paper is to find a mechanism that maximizes the total PV generation revenue $Rev_T$:

$$\text{Objective function:} \quad Rev_T = \max \sum_{i=1}^{T} Rev_i, \tag{1}$$

where $T$ denotes the latest online period, and $Rev_i$ represents the selling revenue of energy produced in period $i$.

This study addresses the challenge of maximizing profitability in photovoltaic (PV) power generation by considering real-time market price fluctuations and constant battery storage loss, denoted as $C_{\text{loss}}$. In this model, PV generation is divided into discrete periods, allowing for strategic management of energy sales. The online algorithm designed to maximize revenue from selling PV power produced in each period. The algorithm strategically reserves at least $\frac{1}{\log h+1}$ of energy for each expected electricity unit price, optimizing revenue by ensuring higher profits are captured at favorable price periods.

In this paper, Sect. 2 describes the notations and models the online trading scenario. Section 3 reveals the One Way Trading algorithm, and Sect. 4 analyzes the performance of the algorithm by computing its competitive ratio and also proving the lower bound of the trading problem. In Sect. 5, the paper is concluded with a discussion of potential research directions in the future.

## 2 Preliminaries

The unit electricity selling prices differ among periods, fluctuating between the lowest $l$ and the highest $h$. We set $l = 2^0 = 1$ and $h = 2^{\log h}$, and assume that any unit price for period $i$ satisfies $R_i \in [2^0, 2^{\log h}]$. For each period $i$, the PV output $P_i$ is divided into $\log h + 1$ shares, with each share containing $\frac{1}{\log h+1} \cdot P_i$ shares of electricity, which can be sold in the current period or later.

**Definition 1.** *For each share $j$ in $0, \cdots, \log h$, an expected selling price is defined as $R^j_{\exp} = 2^j$, which means that these $\frac{1}{\log h+1} \cdot P_i$ shares of electricity in share $j$ are expected to be sold at a unit price of at least $2^j$.*

The shares will be stored and wait to be sold until the unit price reaches the expected price $R^j_{\exp}$, with a constant battery storage loss ratio $C_{loss}$.

**Definition 2.** *The battery storage loss ratio is defined as a constant $C_{loss} \in (0, 1)$, which continuously diminishes the stored energy over time. For example, if the electricity has been stored for t periods, the remaining energy is diminished to $(1 - C_{loss})^t$ of the original amount.*

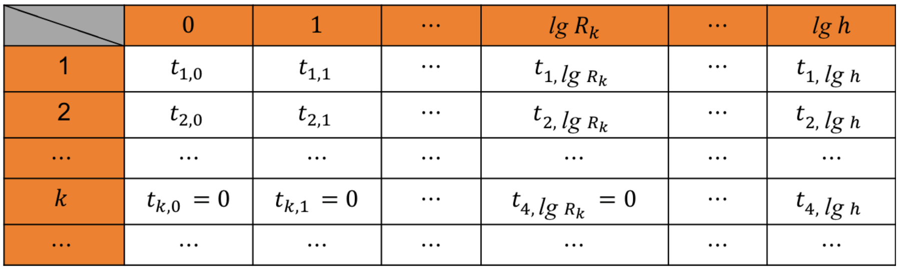
However, the unit selling prices $R_i$ are revealed online and can only be known at the beginning of period $i$. It cannot be known in advance when the expected price will occur, which may lead to significant battery storage loss if waiting too many periods for the high price. Thus, a threshold matrix $T = t_{i,j}$ is defined to control the waiting time, indicating the maximum number of periods that could be waited for the electricity share $j$ in period $i$ to reach the selling price $R^j_{\exp}$ (see Fig. 1).

Since the battery storage loss $C_{loss}$ is constant, and $R^j_{exp}$ is well-defined, $T = t_{i,j}$ can be computed in advance. This allows the threshold matrix to be computed immediately upon the arrival of period $i$. $t_{i,j}$ is computed as follows:

$$R^j_{exp} \cdot (1 - C_{loss})^{t_{i,j}} \cdot \frac{1}{\log h + 1} \cdot P_i = R_i \cdot \frac{1}{\log h + 1} \cdot P_i \tag{2}$$

The left side of Eq. 2 expresses the revenue for selling share $j$ at some future period when the unit price reaches $R^j_{exp}$, considering battery storage loss $(1 - C_{loss})^{t_{i,j}}$. The right side of Eq. 2 indicates the revenue for selling share $j$ in the current period $i$, when the electricity is just produced, without storage loss. The goal of Eq. 2 is to find the period threshold $t_{i,j}$, representing the maximum time to wait for the expected price. Exceeding $t_{i,j}$ means the revenue from a high unit price would be less than selling immediately.

$$t_{i,j} = \begin{cases} \log_{1-C_{\text{loss}}} \left( \frac{R_i}{R^j_{\exp}} \right), & \text{if } R^j_{\exp} \geq R_i, \\ 0, & \text{if } R^j_{\exp} < R_i. \end{cases} \tag{3}$$



| | $0$ | $1$ | $\cdots$ | $lg\ R_k$ | $\cdots$ | $lg\ h$ |
|---|---|---|---|---|---|---|
| $1$ | $t_{1,0}$ | $t_{1,1}$ | $\cdots$ | $t_{1,\,lg\ R_k}$ | $\cdots$ | $t_{1,\,lg\ h}$ |
| $2$ | $t_{2,0}$ | $t_{2,1}$ | $\cdots$ | $t_{2,\,lg\ R_k}$ | $\cdots$ | $t_{2,\,lg\ h}$ |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $k$ | $t_{k,0} = 0$ | $t_{k,1} = 0$ | $\cdots$ | $t_{4,\,lg\ R_k} = 0$ | $\cdots$ | $t_{4,\,lg\ h}$ |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |

**Fig. 1.** The threshold matrix $T$ stores values $t_{ij}$ for each share $j$ in period $i$, indicating the maximum number of periods that can be waited for the electricity share $j$ in period $i$ to reach the selling price $R^j_{\text{exp}}$. For shares $j \leq \log R_k$ in any period $k$, $t_{k,j}$ is set to zero, as these shares are sold immediately without delay. For other shares, $t_{i,j}$ is computed using formula 3.

Equation 3 computes the exact $t_{i,j}$ according to Eq. 2. This allows the threshold matrix to be computed immediately upon the arrival of period $i$. (see Fig. 1). In this paper, revenue is analysed by periods:

$$\max Rev_i = \max \sum_j Rev_{i,j} \tag{4}$$

where $Rev_{i,j}$ indicates the selling revenue of electricity share $j$ produced in period $i$.

## 3 One Way Trading Algorithm

In this section, the Algorithm 1 for One Way Trading on the PV generation is revealed. As illustrated in Fig. 2, for each period $i$, the produced PV energy is allocated and potentially sold across multiple periods. Shares are held until either their expected selling price $R^j_{\text{exp}}$ is met or the maximum waiting time defined by $i + t_{i,j}$ is reached. If a share $j$ has not been sold by the time $i + t_{i,j}$ arrives, it will be sold at that period's price to avoid further battery storage loss.

**Description of One Way Trading.** Algorithm 1 takes the highest unit selling price $h$ and the constant battery storage loss ratio $C_{\text{loss}}$ as inputs, which are assumed to be known in advance, and outputs the total revenue $Rev_i$ of period $i$, which is the target to be maximized (see objective function 1). In initialization, the threshold matrix $T$ is computed by formula 3. When period $i$ comes, the PV output surplus $P_i$ and unit electricity price $R_i$ are revealed immediately. $P_i$ is divided into $\log h + 1$ shares, with each share $j$'s revenue $Rev_{i,j}$ initialized to zero.

$Rev_i$ consists of two parts: the first is the part immediately sold in period $i$, which sells $\log R_i$ shares of electricity, with revenue

$$Rev_i = R_i \cdot \frac{1}{\log h + 1} \cdot P_i \cdot \lfloor \log R_i + 1 \rfloor. \tag{5}$$

**Algorithm 1.** One Way Trading

```
Input: The highest unit price $h$, the battery storage loss ratio $C_{\text{loss}}$;
Output: The total revenue $Rev_i$ of period $i$;
 1: Initialize: Compute the threshold matrix $T$;
 2: Get $P_i$ and $R_i$ at the beginning of period $i$, set $Rev_i = 0$;
 3: Divide $P_i$ into $\log h + 1$ shares, initialize each share $j$'s revenue $Rev_{i,j} = 0$;
 4: $Rev_i = R_i \cdot \frac{1}{\log h+1} \cdot P_i \cdot (\lfloor \log R_i \rfloor + 1)$;
 5: for $j = \lfloor \log R_i + 1 \rfloor, \ldots, \log h$ do
 6:     for period $k = i+1, \ldots, i + t_{i,j}$ do
 7:         if $R_k \geq R^j_{\text{exp}}$ then
 8:             Sell share $j$, set $Rev_{i,j} = R_k \cdot \frac{1}{\log h+1} \cdot P_i \cdot (1 - C_{\text{loss}})^{(k-i)}$;
 9:             break;
10:         else if $k = i + t_{i,j}$ then
11:             Sell share $j$ at the current period $k$'s price;
12:             $Rev_{i,j} = R_k \cdot \frac{1}{\log h+1} \cdot P_i \cdot (1 - C_{\text{loss}})^{t_{i,j}}$;
13:         end if
14:     end for
15:     $Rev_i$ += $Rev_{i,j}$;
16: end for
17: return The total revenue $Rev_i$ of period $i$.
```

The second part is sold in the following periods $k$, whose revenue depends on the unit price $R_k$. For each $k$, the corresponding revenue is

$$Rev_{i,j} = R_k \cdot \frac{1}{\log h + 1} \cdot P_i \cdot (1 - C_{\text{loss}})^{(k-i)}. \tag{6}$$

In the first **for** loop, $j$ traverses all the shares from $\lfloor \log R_i + 1 \rfloor$ to $\log h$ (as the shares $j \leq \lfloor \log R_i + 1 \rfloor$ are already sold, they should not be traversed), to compute each share $j$'s revenue $Rev_{i,j}$ and accumulate it to the current $Rev_i$. In the second **for** loop, period $k$ traverses from $i + 1$ to $i + t_{i,j}$, to determine the proper period to sell share $j$. If some period $k$'s unit price $R_k$ reaches $R^j_{\text{exp}}$, sell the share $j$ and the inner **for** loop breaks. As the share $j$ is sold at that period $k$, there is no longer a need to continue checking future periods for that share since the decision has already been made. However, if share $j$ is not sold until the end of $k$'s traversal, it will be sold at period $k = i + t_{i,j}$ with whatever the current price $R_k$ is, considering too much waiting might bring great loss caused by $C_{\text{loss}}$, and the revenue is

$$Rev_{i,j} = R_k \cdot \frac{1}{\log h + 1} \cdot P_i \cdot (1 - C_{\text{loss}})^{t_{i,j}}. \tag{7}$$

At the end of Algorithm 1, $Rev_i$ is returned.

## 4 Performance Analysis

In this section, we first analyze the competitive ratio of Algorithm 1, which quantifies the algorithm's performance relative to an optimal offline algorithm.

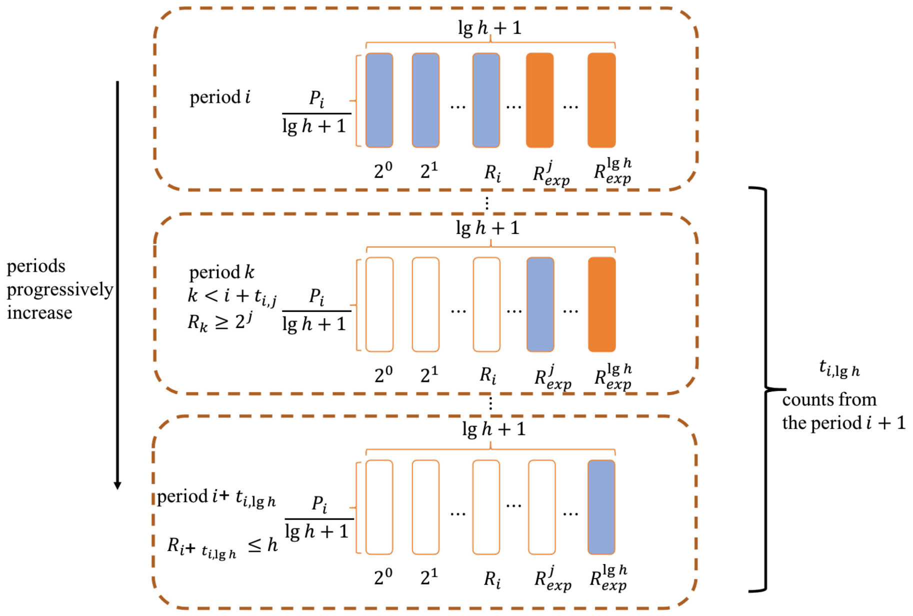

lg $h$ + 1
period $i$
$P_i$ / lg $h$ + 1
$2^0$ $2^1$ $R_i$ $R^j_{exp}$ $R^{\lg h}_{exp}$
periods progressively increase
period $k$
$k < i + t_{i,j}$
$R_k \geq 2^j$
period $i$+ $t_{i,\lg h}$
$R_{i+t_{i,\lg h}} \leq h$
$t_{i,\lg h}$ counts from the period $i+1$

**Fig. 2.** Periods are depicted by dashed boxes in the figure. Blue columns represent energy shares sold in the current period, while orange columns indicate those held for future sale; blank columns denote shares sold in past periods. The numbers under the columns correspond to the expected unit prices. In period $i$, PV energy is allocated and potentially sold across multiple periods. Shares indexed by $j \leq \log R_i$ are sold within period $i$ itself. For shares where $j > \log R_i$, they are designated to be sold in a future period $k$ where $R_k \geq R^j_{\text{exp}}$, provided that $k \leq i + t_{i,j}$. Any share $j$ must be sold by the threshold period $i + t_{i,j}$, ensuring no further battery storage loss occurs. (Color figure online)

Subsequently, we explore the theoretical lower bound of the trading problem to establish a baseline for comparison. It is demonstrated that Algorithm 1, referred to as One Way Trading, achieves a competitive ratio that matches the problem's lower bound, establishing its optimality at $\log h$.

## 4.1 Competitive Ratio

The competitive ratio of an online algorithm is defined as the worst-case ratio of the revenue obtained by an optimal offline algorithm (OPT) to the revenue obtained by the online algorithm. For Algorithm 1, this ratio is expressed as:

$$\alpha = \frac{\text{Revenue of OPT}}{\text{Revenue of Algorithm 1}} \tag{8}$$

This measure reflects how closely the online algorithm's performance approximates the best possible outcome achievable by any algorithm with complete future knowledge.

**Lemma 1.** *The OPT algorithm will sell all the $P_i$ electricity before period $i + t_{i,\log h}$, with revenue given by*

$$Rev_i^{opt} = R_{i+opt} \cdot P_i \cdot (1 - C_{loss})^{opt}, \tag{9}$$

*where opt denotes the periods interval between period $i$ and the optimal selling period $i + opt$, and $opt \leq t_{i,\log h}$.*

*Proof.* Lemma 1 is proved by combining Propositions 1 and 2. The OPT algorithm's revenue is computed by formula 9, which naturally follows from these propositions. □

**Proposition 1.** *The OPT algorithm sells all the $P_i$ electricity at once at a sufficiently high unit price.*

*Proof.* To maximize the total revenue $Rev_i$, the OPT algorithm sells all the electricity produced in period $i$ at once. As an offline algorithm, OPT has complete knowledge of future unit prices and can choose the optimal selling period that balances unit price and waiting time to achieve maximal revenue. Once such a period is chosen, there is no advantage in holding any electricity; therefore, all of it should be sold immediately.

Consider a scenario with two possible selling periods: an earlier period with a relatively low unit price and a later period with a relatively high unit price, where the total revenue from both periods would be the same due to increased battery storage loss in the latter. In this situation, OPT can choose either period to sell all the $P_i$ electricity, as the revenue will be maximized in both scenarios.

Thus, OPT ensures that all electricity is sold at once in the period where the expected revenue is highest. □

**Proposition 2.** *The period $i + opt$ chosen by OPT to sell $P_i$ is earlier than the arrival of the maximal waiting threshold $i + t_{i,\log h}$.*

*Proof.* As indicated by formula 2, the threshold $t_{i,j}$ represents the maximum number of periods that could be waited for a high unit price while avoiding significant battery storage loss. The value $t_{i,\log h}$ corresponds to the maximal threshold, implying that any period after $i + t_{i,\log h}$ with a high unit price (given that the unit price cannot exceed $h$) results in less revenue than selling $P_i$ immediately at period $i$.

Since the OPT algorithm has full foresight of future prices, it selects period $i + \text{opt}$ to sell $P_i$ before reaching $i + t_{i,\log h}$, ensuring maximum revenue without incurring excessive storage losses. Thus, the chosen period by OPT is guaranteed to precede the maximal waiting threshold. □

**Lemma 2.** *Algorithm 1 achieves a revenue $Rev_i^{alg}$ such that*

$$Rev_i^{alg} \geq \frac{1}{\log h + 1} \cdot Rev_i^{opt}, \tag{10}$$

*where $Rev_i^{alg}$ denotes the revenue achieved by the algorithm, and $Rev_i^{opt}$ is the revenue achieved by the OPT algorithm.*

*Proof.* Algorithm 1 reserves $\frac{1}{\log h+1} \cdot P_i$ of the electricity for each possible unit price $R^j_{\text{exp}} \in \{2^0, 2^1, \cdots, h\}$ in period $i$, until the corresponding threshold is reached. According to Proposition 2, the OPT algorithm will sell all of $P_i$ at period $i + \text{opt}$, which occurs before the maximal threshold period $i + t_{i,\log h}$.

Algorithm 1 will also evaluate period $i + \text{opt}$ and sell at least $\frac{1}{\log h+1} \cdot P_i$ at this period with a revenue of

$$\begin{aligned} Rev_i^{\text{alg}} &\geq R_{i+\text{opt}} \cdot \frac{1}{\log h + 1} \cdot P_i \cdot (1 - C_{\text{loss}})^{\text{opt}} \\ &= \frac{1}{\log h + 1} \cdot Rev_i^{\text{opt}}. \end{aligned} \tag{11}$$

It is important to note that $R_{i+\text{opt}}$ must have reached the expected unit price $R^j_{\text{exp}}$ of the electricity shares $j$ that Algorithm 1 sells at the $i+\text{opt}$ period. Otherwise, the OPT algorithm would not choose this period, as it would be no better than selling the electricity at period $i$. □

**Theorem 1.** *Given the highest unit price $h$, Algorithm 1 achieves a competitive ratio of $\alpha = O(\log h)$ for the photovoltaic generation trading problem.*

*Proof.* By Lemma 1 and Lemma 2, for each period $i$:

$$\frac{Rev_i^{\text{opt}}}{Rev_i^{\text{alg}}} = O(\log h). \tag{12}$$

Then, summing up the revenues from $i = 1$ to $i = T$ and by local ratio lemma:

$$\alpha = \frac{Rev_T^{\text{opt}}}{Rev_T^{\text{alg}}} = \frac{\sum_{i=1}^{T} Rev_i^{\text{opt}}}{\sum_{i=1}^{T} Rev_i^{\text{alg}}} \leq \max_i \frac{Rev_i^{opt}}{Rev_i^{alg}} \leq O(\log h), \tag{13}$$

where $Rev_T^{\text{opt}}$ and $Rev_T^{\text{alg}}$ denote the total revenue over all periods achieved by the OPT and Algorithm 1, respectively. □

## 4.2 Lower Bound

In online algorithms, the lower bound of a problem refers to the best possible performance that any algorithm can achieve under the given conditions. This bound is derived from the inherent uncertainties in price fluctuations and storage capabilities. Assume there is an adversary that creates a challenging environment for any algorithm to perform well, leading to a situation where the solution to the problem can never exceed this bound, known as the lower bound of the problem. In this trading problem, an adversary is defined as follows:

**Adversary.** During trading, the unit price for each period is revealed online. The adversary determines a price distribution $R_{i'}$ across periods $i'$ as:

$$R_{i'} = \frac{2^{i'-i}}{(1 - C_{\text{loss}})^{i'-i}}, R_{i'} \in [l, h]. \tag{14}$$

Define $a_{i,i'}$ as the actual amount of electricity from period $i$ that is sold in period $i'$, where $i' \in [i, i'_{\max}]$. In period $i'$, if $a_{i,i'}$ satisfies the condition:

$$\frac{a_{i,i'}}{(1 - C_{\text{loss}})^{i'-i}} \geq \frac{1}{\log h + 1} \cdot P_i, \tag{15}$$

the adversary continues moving to the next period and offering prices. Otherwise, the adversary stops the process.

Note that $\frac{a_{i,i'}}{(1-C_{\text{loss}})^{i'-i}}$ computes the corresponding share from the original period $i$, taking into account battery storage loss.

**Lemma 3.** *The adversary will stop at the period* $i'_{\max}$.

*Proof.* For all $i' < i'_{\max}$, the condition in Eq. 15 holds. Since there is a total of $P_i$ electricity to be sold:

$$\sum_{i'=i}^{i'_{\max}} \frac{a_{i,i'}}{(1 - C_{\text{loss}})^{i'-i}} \leq P_i, \tag{16}$$

there must be one period $i'_{\max}$ such that

$$\frac{a_{i,i'_{\max}}}{(1 - C_{\text{loss}})^{i'_{\max}-i}} < \frac{1}{\log h + 1} \cdot P_i. \tag{17}$$

Thus, the adversary will stop at this point. □

**OPT Performance on Adversary.** The price offered by the adversary grows faster than the rate of battery storage loss, making it optimal for the OPT algorithm to sell all of $P_i$ at the last possible period, denoted as $i'_{\max}$. The revenue obtained by the OPT algorithm in this scenario $Rev_{opt'}$ is given by:

$$\begin{aligned} Rev_{opt'} &= \frac{2^{i'_{\max}-i}}{(1 - C_{\text{loss}})^{i'_{\max}-i}} \cdot (1 - C_{\text{loss}})^{i'_{\max}-i} \cdot P_i \\ &= 2^{i'_{\max}-i} \cdot P_i. \end{aligned} \tag{18}$$

**Any Online Algorithm Performance on Adversary.** Denote by $Rev_{alg'}$ the revenue that any algorithm could achieve under the adversary:

$$\begin{aligned} Rev_{alg'} &= \sum_{i'=i}^{i'_{\max}-1} \frac{2^{i'-i}}{(1 - C_{\text{loss}})^{i'-i}} \cdot a_{i,i'} + \frac{2^{i'_{\max}-i}}{(1 - C_{\text{loss}})^{i'_{\max}-i}} \cdot a_{i,i'_{\max}} \\ &\leq 2 \cdot \frac{2^{i'_{\max}-i}}{(1 - C_{\text{loss}})^{i'_{\max}-i}} \cdot a_{i,i'_{\max}}, \end{aligned} \tag{19}$$

with

$$a_{i,i'_{\max}} \leq \frac{(1 - C_{\text{loss}})^{i'_{\max}-i}}{\log h + 1} \cdot P_i. \tag{20}$$

Then the revenue satisfies:

$$\begin{aligned} Rev_{alg'} &\leq 2 \cdot \frac{2^{i'_{\max}-i}}{(1 - C_{\text{loss}})^{i'_{\max}-i}} \cdot \frac{(1 - C_{\text{loss}})^{i'_{\max}-i}}{\log h + 1} \cdot P_i \\ &= 2 \cdot \frac{2^{i'_{\max}-i}}{\log h + 1} \cdot P_i. \end{aligned} \tag{21}$$

**Theorem 2.** *The lower bound of the trading problem addressed by Algorithm 1, is $\gamma = \Omega(\log h)$.*

*Proof.* By the algorithm performance analysis, it can be computed that the lower bound:

$$\gamma \geq \frac{Rev_{opt'}}{Rev_{alg'}} = \frac{\log h + 1}{2} = \Omega(\log h). \tag{22}$$

□

This matching of the competitive ratio and the lower bound confirms that Algorithm 1 is tight within the modeled trading environments.

## 5 Conclusion and Discussion

This work focuses on one-way trading by considering the sale of energy produced by PV generation for profit. The algorithm proposed in this study optimizes the selling strategy to maximize revenue from PV power sales. However, this approach does not consider the potential need for energy storage or the purchase of electricity from the grid during periods of low PV output, such as rainy days or at night. PV holders often face the decision of whether to store surplus electricity for later use or sell it to the grid immediately to avoid storage losses. This introduces the two-way trading, where both selling and buying decisions are considered to optimize overall benefits. Future research could explore the two-way trading problem, where the algorithm not only maximizes profits from selling but also minimizes costs associated with purchasing electricity from the grid.

**Acknowledgments.** This study was funded by National Key R&D Program of China (No. 2022YFE0196100), Guangdong Basic and Applied Basic Research Foundation 2024A1515030197, NSFC 12071460, NSFC 12371321, Shenzhen Science and Technology Program CJGJZD20210408092806017, Shenzhen Polytechnic Research Fund No. 6023310009K.

## References

1. Alinia, B., Hajiesmaili, M.H., Lee, Z.J., Crespi, N., Mallada, E.: Online ev scheduling algorithms for adaptive charging networks with global peak constraints. IEEE Trans. Sustain. Comput. **7**(3), 537–548 (2022)
2. Balcan, M.F., Blum, A.: Approximation algorithms and online mechanisms for item pricing. In: Proceedings of the 7th ACM Conference on Electronic Commerce, pp. 29–35 (2006)
3. Bansal, N., Chen, N., Cherniavsky, N., Rurda, A., Schieber, B., Sviridenko, M.: Dynamic pricing for impatient bidders. ACM Trans. Algor. (TALG) **6**, 1–21 (2010)
4. Blum, A., Kumar, V., Rudra, A., Wu, F.: Online learning in online auctions. Theor. Comput. Sci. **324**(2–3), 137–146 (2004)
5. Blum, A., Sandholm, T., Zinkevich, M.: Online algorithms for market clearing. J. ACM (JACM) **53**(5), 845–879 (2006)
6. Briest, P.: Uniform budgets and the envy-free pricing problem. In: International Colloquium on Automata, Languages, and Programming, pp. 808–819 (2008)
7. Briest, P., Krysta, P.: Buying cheap is expensive: hardness of non-parametric multi-product pricing. In: SODA, pp. 716–725 (2007)
8. Chen, N., Deng, X.: Envy-free pricing in multi-item markets. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 418–429. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14162-1_35
9. Chin, F.Y., et al.: Competitive algorithms for unbounded one-way trading. Theor. Comput. Sci. **607**, 35–48 (2015)
10. Ferreira, K.J., Simchi-Levi, D., Wang, H.: Online network revenue management using Thompson sampling. Oper. Res. **66**(6), 1586–1602 (2018)
11. Guo, X., Dai, S., Han, X., Shang, M., Xu, Y., Zhang, Y.: Trade-off between maximum flow time and energy intake in ev charging. In: To appear in International Computing and Combinatorics Conference, COCOON 2024, Shanghai, China, 23–25 August 2024 (2024)
12. Ha, Y., Zhang, H.: Algorithmic trading for online portfolio selection under limited market liquidity. Eur. J. Oper. Res. **286**(3), 1033–1051 (2020)
13. Han, L., Wang, C., Xu, D., Zhang, D.: Algorithms for the prize-collecting $k$-steiner tree problem. Tsinghua Sci. Technol. **27**(5), 785–792 (2022)
14. Mulder, G., Six, D., Claessens, B., Broes, T., Omar, N., Van Mierlo, J.: The dimensioning of pv-battery systems depending on the incentive and selling price conditions. Appl. Energy **111**, 1126–1135 (2013)
15. Ru, Y., Kleissl, J., Martinez, S.: Storage size determination for grid-connected photovoltaic systems. IEEE Trans. Sustain. Energy **4**, 68–81 (2013)
16. Tan, J., Sun, Y., Xu, Y., Zou, J.: Streaming algorithms for non-submodular maximization on the integer lattice. Tsinghua Sci. Technol. **28**(5), 888–895 (2023)
17. Xie, J., Shugan, S.M.: Electronic tickets, smart cards, and online prepayments: when and how to advance sell. Mark. Sci. **20**(3), 219–243 (2001)
18. Xin, H., Liu, Y., Wang, Z., Gan, D., Yang, T.: A new frequency regulation strategy for photovoltaic systems without energy storage. IEEE Trans. Sustain. Energy **4**, 985–993 (2013)
19. Yang, R., Gao, S., Han, L., Li, G., Zhao, Z.: Approximating $(m_B, m_P)$-monotone bp maximization and extensions. Tsinghua Sci. Technol. **28**(5), 906–915 (2023)
20. Zhang, W., Xu, Y., Zheng, F., Dong, Y.: Optimal algorithms for online time series search and one-way trading with interrelated prices. J. Comb. Optim. **23**(2), 159–166 (2012)

21. Zhang, Y., Chen, J., Chin, F., Han, X., Ting, H.-F., Tsin, Y.H.: Improved online algorithms for 1-space bounded 2-dimensional bin packing. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6507, pp. 242–253. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17514-5_21
22. Zhang, Y., Chin, F.Y., Ting, H.F.: Online pricing for bundles of multiple items. J. Glob. Optim. **58**, 377–387 (2014)

# Parameterized Complexity of Shortest Path with Positive Disjunctive Constraints

Susobhan Bandopadhyay[1], Suman Banerjee[2], Diptapriyo Majumdar[3](✉), and Fahad Panolan[4]

[1] National Institute of Science Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, Odisha, India
susobhan.bandopadhyay@niser.ac.in

[2] Department of Computer Science and Engineering, Indian Institute of Technology, Jammu, India
suman.banerjee@iitjammu.ac.in

[3] Indraprastha Institute of Information Technology Delhi, New Delhi, India
diptapriyo@iiitd.ac.in

[4] School of Computer Science, University of Leeds, Leeds, UK
F.Panolan@leeds.ac.uk

**Abstract.** We study the Shortest Path problem subject to positive binary disjunctive constraints. In positive disjunctive constraints, there are certain pairs of edges such that at least one edge from every pair must be part of every feasible solution. We initiate the study of Shortest Path with binary positive disjunctive constrains from the perspective of parameterized complexity. Formally, the input instance is a simple unidirected graph $G = (V, E)$, a forcing graph $G_f = (E, E')$, two vertices $s, t \in V(G)$ and an integer $k$. Note that the vertex set of $G_f$ is the same as the edge set of $G$. The goal is to find a set $S$ of at most $k$ edges from $G$ such that there is a path from $s$ to $t$ in the subgraph $G = (V, S)$ and $S$ is a vertex cover in $G_f$. In this paper, we consider two different natural parameterizations for this problem. One natural parameter is the solution size, i.e. $k$ for which we provide FPT algorithms and polynomial kernelization results. The other natural parameters are structural parameterisations of $G_f$, i.e. the size of a modulator $X \subseteq E(G) = V(G_f)$ such that $G_f - X$ belongs to some hereditary graph class. We discuss the parameterized complexity of this problem under some structural parameterizations.

**Keywords:** Shortest Path · Parameterized Complexity · Positive Disjunctive Constraints · Kernelization · Planar Graph

## 1 Introduction

In the recent times several classical combinatorial optimization problems on graphs including Maximum Matching, Shortest Path, Steiner Tree have

Research of Diptapriyo Majumdar has been supported by Science and Engineering Research Board (SERB) grant SRG/2023/001592.

been studied along with some additional binary conjunctive and/or disjunctive constraints [1,7]. Darmann et al. [7] have studied finding shortest paths, minimum spanning trees, and maximum matching of graphs with binary disjunctive constraints in the perspective of classical complexity. These positive or negative binary constraints are defined with respect to pairs of edges. A *negative disjunctive constraint* between an edge-pair $e_i$ and $e_j$ says that both $e_i$ and $e_j$ cannot be present in a feasible solution. A *positive disjunctive constraint* between an edge pair $e_i$ and $e_j$ says that either the edge $e_i$ or the edge $e_j$ or both must be present in any feasible solution. The negative disjunctive constraints can be interpreted as a *conflict graph* such that each vertex of the conflict graph corresponds to an edge in the original graph. Furthermore, for every edge in the conflict graph, at least one endpoint can be part of any feasible solution. Then, a feasible solution must be an independent set in the conflict graph. The positive disjunctive constraints can be interpreted as a *forcing graph* such that each vertex of the forcing graph corresponds to an edge in the original graph. In the forcing graph, each edge must have at least one endpoint included in any feasible solution. Therefore, in the case of positive disjunctive constraints, a feasible solution must be a vertex cover in the forcing graph. Formally, an input to the Forcing-Version of a classical combinatorial optimization problem $\Pi$, called as Forcing-Version $\Pi$ consists of an instance $I$ of $\Pi$ along with a forcing graph $G_f$; i.e.; $(I, G_f)$. The vertex set of the forcing graph is the edge set of the original graph. A solution of Forcing-Version $\Pi$ for the instance $(I, G_f)$ is a solution of $I$ for the original problem along with the property that the solution forms a vertex cover for $G_f$. To the best of our knowledge, none of the problems Shortest Path, Maximum Matching, Minimum Spanning Tree have been explored with positive disjunctive constraints in the perspective of parameterized complexity.

In this paper, we initiate the study of Shortest Path problem with positive disjunctive constraints from the perspective of parameterized complexity and kernelization (see Sect. 2 for definitions etc.). Formally, in the Shortest Path with Forcing Graph (SPFG) problem, we are given a simple, unweighted graph $G(V, E)$, two vertices $s$ and $t$, a positive integer $k$ and a forcing graph $G_f(E, E^{'})$. The decision version of this problem asks to check whether there exists a set $E^* \subseteq E(G)$ of at most $k$ edges such that the subgraph induced by the vertex set $V(E^*)$ in $G$ contains an $s$-$t$ path and also $E^*$ forms a vertex cover in $G_f$. As "solution size" is the most natural parameter, we formally state the definition of the parameterized version of our problem as follows.

Shortest Path with Forcing Graph (SPFG) **Parameter:** k
**Input:** A simple, undirected graph $G(V, E)$, two distinct vertices $s, t \in V(G)$, a positive integer $k$, and a forcing graph $G_f(E, E^{'})$.
**Question:** Is there a set $E^*$ of at most $k$ edges from $G$ such that the subgraph $G(V, E^*)$ contains an $s$-$t$ path in $G$, and $E^*$ forms a vertex cover in $G_f$?

In the above definition, the considered parameter is 'solution size'. Darmann et al. [7] have proved that the classical version of Shortest Path with Forcing Graph is NP-hard even when the forcing graph $G_f$ is a graph of degree at most one. So, it can be concluded that even when the forcing graph $G_f$ is very sparse, then also the classical version of Shortest Path with Forcing Graph is NP-hard. In the first part of our paper, we consider Shortest Path with Forcing Graph when parameterized by solution size. Additionally, it is also natural to consider some parameters that are some structures of the input. Observe that the solution size must be as large as the minimum vertex cover size of the forcing graph. If we consider the 'deletion distance of $G_f$ to some hereditary graph class $\mathcal{G}$', then this deletion distance is a parameter that is provably smaller than solution size whenever $\mathcal{G}$ contains a graph that has at least one edge. In the second part of our paper, we also initiate the study of this problem when the considered parameter is deletion distance of $G_f$ to some special graph class. Formally, the definition of this parameterized version is the following.

SPFG-$\mathcal{G}$-Deletion **Parameter:** $|X|$
**Input:** A simple, undirected graph $G(V,E)$, two distinct vertices $s,t \in V(G)$, a positive integer $k$, a forcing graph $G_f(E,E^{'})$, a set $X \subseteq E$ such that $G_f - X \in \mathcal{G}$.
**Question:** Is there a subset $E^*$ of at most $k$ edges from $G$ such that the subgraph $G(V,E^*)$ contains an $s$-$t$ path in $G$, and $E^*$ forms a vertex cover in $G_f$?

*Our Contributions:* In this paper, we study the Shortest Path with Forcing Graph under the realm of parameterized complexity and kernelization (i.e. polynomial-time preprocessing). We first consider both the original graph $G$ and the forcing graph $G_f$ to be arbitrary graphs and provide FPT and kernelization algorithms. Next, we initiate a systematic study on what happens to the kernelization complexity when either $G$ is a special graph class or $G_f$ is a special graph class. Formally, we provide the following results for Shortest Path with Forcing Graph when the solution size ($k$) is considered as the parameter.

➢ First, we prove two preliminary results. One preliminary result is a polynomial time algorithm for SPFG when $G_f$ is $2K_2$-free (see Lemma 2). Implication of this result is a dichotomy that SPFG is polynomial-time solvable when $G_f$ is the class of all $2K_2$-free and NP-hard otherwise (see Theorem 1). The other preliminary result is a parameterized algorithm for Shortest Path with Forcing Graph that runs in $\mathcal{O}^*(2^k)$-time[1] (see Theorem 2).

➢ Then, we prove our main result. In particular, we prove that Shortest Path with Forcing Graph admits a kernel with $\mathcal{O}(k^5)$-vertices when $G$ and $G_f$ both are arbitrary graphs (see Theorem 3).

➢ After that, we consider the kernelization complexity of Shortest Path with Forcing Graph when $G$ is a planar graph and $G_f$ is an arbitrary graph. In this condition, we provide a kernel with $\mathcal{O}(k^3)$ vertices (see Theorem 4).

➢ Next, we consider when $G$ is an arbitrary graph and $G_f$ is a graph belonging to a special hereditary graph class. In this paper, we focus on the condition

[1] $\mathcal{O}^*$ hides polynomial factor in the input size.

when $G_f$ is a cluster graph (i.e. a disjoint union of cliques) or a bounded degree graph. In both these conditions, we provide a kernel with $\mathcal{O}(k^3)$ vertices for Shortest Path with Forcing Graph (see Theorem 5).

Finally, we consider the SPFG-$\mathcal{G}$-Deletion problem. It follows from the results of Darmann et al. [7] that even if $X = \emptyset$ and $G_f$ is a 2-ladder, i.e. a graph of degree one, SPFG-$\mathcal{G}$-Deletion is NP-hard. Therefore, it is unlikely to expect the possibility that SPFG-$\mathcal{G}$-Deletion would admit an FPT algorithm even when $\mathcal{G}$ is a very sparse graph classes. We complement their NP-hardness result by proving that SPFG-$\mathcal{G}$-Deletion admits an FPT algorithm when $\mathcal{G}$ is the class of all $2K_2$-free graphs (see Theorem 6).

*Related Work:* Recently, conflict free and forcing variant of several classical combinatorial optimization problem including Maximum Flow [14], Maximum Matching [6], Minimum Spanning Tree [6,7], Set Cover [10], Shortest Path [7] etc. have been studied extensively in both algorithmic and complexity theoretic point of view. Recently, some of these problems also have been studied in the realm of parameterized complexity as well [1,12]. Agrawal et al. [1] have studied Shortest Path and Maximum Matching with conflict free version and proved that both the problems are W[1]-hard when parameterized by solution size. They also investigated the complexity of the problems when the conflict graph has some topological structure. Darmann et al. [7] studied both the problems along with both the constraints conflict graph and forcing graph. they showed that the conflict free variant of maximum matching problem is NP-hard even when the conflict graph is a collection of disjoint edges.

*Organization of the Paper:* We organize our paper as follows. In Sect. 2, we provide some notations related to graph theory, and parameterized complexity. In the same section, we also prove our first two preliminary results (Theorem 1 and Theorem 2). After that, in Sect. 3, we prove the main result (Theorem 3) of our paper. Next, in Sect. 4, we give a short illustration how we can improve the size of our kernels of Sect. 3 when either the input graph $G$ or $G_f$ belongs to some special graph classes. Additionally, in the same section, we provide a result (Theorem 6) of SPFG on the structural parameterizations for SPFG. Finally, in Sect. 5, we conclude with open problems and future research directions.

## 2 Preliminaries

In this section, we describe the notations and symbols used in this paper. Additionally, we also provide some preliminary results for our porblem in this section.

*Graph Theory:* All the graphs considered in this paper are simple, finite, undirected and unweighted. The notations and terminologies used in this paper are fairly standard and adopted from the Diestel's book of graph theory [8]. In our problem, we are dealing with two different graphs: original graph $G$ and forcing graph $G_f$. We denote the number of vertices and edges of $G$ by $n$ and $m$, respectively. Similarly for $G_f$, it is $m$ and $m'$ (since the edge set of $G$ is same as the vertex set of $G_f$). Given a graph $G(V, E)$ and an edge $e \in E(G)$, $V_e$ denotes the set of two end vertices of $e$. For any subset of edges $E^* \subseteq E(G)$, by $V(E^*)$ we denote the set of all the vertices that constitute the edge set; i.e.; $V(E^*) = \bigcup\limits_{e \in E^*} V_e$. Informally, $V(E^*)$ denote the set of all endpoints of the edges in $E^*$. Given, any two vertices $u$ and $v$, we denote its shortest path distance by $dist(u, v)$. For any graph $G(V, E)$ a subset of its vertices $S \subseteq V(G)$ is said to be a *vertex cover* of $G$ if every edge of $G$ has at least one of its endpoints in $S$. A subset of the vertices $S$ is said to be an independent set of $G$ if between any pair of vertices of $S$, there does not exist any edge in $G$. For any subset of vertices $S$ of $G$, the subgraph induced by the vertex set in $G$ is denoted by $G[S]$. Furthermore, for a set of edges $F$, the graph $G[F]$ is the graph with $G'(V, F)$. Informally, given an edge set $F \subseteq E(G)$, the graph $G[F]$ has vertex set $V(G)$ and the edge set $F$. For any graph $G$ and any vertex $v \in V(G)$, $G - \{v\}$ denotes the graph that can be obtained by deleting $v$ and the edges incident on it from $G$. This notion can be extended for a subset of vertices as well. Given a graph $G = (V, E)$ and a set $X \subseteq V(G)$. A graph operation *identification* of the vertex subset $X$ into a new vertex $u_X$ is performed by constructing a graph $\hat{G}$ as follows. First, delete the vertices of $X$ from $G$ and then add a new vertex $u_X$. Then, for every $v \in N_G(X)$, make $vu_X$ an edge of $\hat{G}$. This graph operation was also defined in Majumdar et al. [13]. A graph is said to be a *cluster graph* if every connected component is a clique. A graph is said to be a *degree-$\eta$-graph* if every vertex has degree at most $\eta$. A connected graph is said to be $2K_2$*-free* if it does not contain any pair of edges that are nonadjacent to each other. A graph is said to be a *planar graph* if it can be drawn in the surface of a sphere without crossing edges. We use the following property of planar graph in our results.

**Proposition 1** ([15]). *If $G$ is a simple planar graph with $n$ vertices, then $G$ has at most $3n - 6$ edges.*

A graph is said to be a 2-*ladder* if every connected component is a path of length one. Similarly, a graph is said to be a 3-*ladder* if every connected component is a path of length two.

*Parameterized Complexity and Kernelization:* A *parameterized problem $\Pi$* is denoted as a subset $\Sigma^* \times \mathbb{N}$. An instance to a parameterized problem is denoted by $(I, k)$ where $(I, k) \in \Sigma^* \times \mathbb{N}$ where $\Sigma$ is a finite set of alphabets and $\mathbb{N}$ is the set of natural numbers. A parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is said to be *fixed-parameter tractable* (or *FPT* in short) if there exists an algorithm $\mathcal{A}$ which runs in $\mathcal{O}(f(k) \cdot |I|^c)$ time where $f(k)$ is a function of $k$ and independent of $n$ and $c$ is a positive constant independent of $n$ and $k$. We denote the running time $\mathcal{O}(f(k) \cdot |I|^c)$ by the shorthand notation $\mathcal{O}^*(f(k))$ where we suppress the polynomial factors. We adopt the notations and symbols related to parameterized algorithms form the books [4,9]. A parameterized problem $\Pi$ admits a *kernelization* (or *kernel* in short) if starting with any arbitrary instance $(I, k)$ of the problem, there exists a polynomial-time algorithm that constructs an equivalent instance $(I^{'}, k^{'})$

such that $|I^{'}| + k^{'} \leq g(k)$ for some commutable function $g(\cdot)$. This function $g(\cdot)$ denotes the *size* of the kernel. If $g(k)$ is bounded by a function polynomial in $k$, then $\Pi$ is said to admit a *polynomial kernel*. It has been shown by Cai et al. [2] that a problem is in FPT if and only if there exists a kernalization. If a parameterized problem $\Pi$ admits a kernelization algorithm, we also call that $\Pi$ admits a *kernel* (in short). We describe the kernalization process by writing a number of reduction rules. A reduction rule takes one instance (say $\mathcal{I}$) of $\Pi$ and generates the reduced instance (say $\mathcal{I}^{'}$) of $\Pi$. We say a reduction rule is *safe* if the following condition holds: "$\mathcal{I}$ is a Yes-instance if and only if $\mathcal{I}^{'}$ is a Yes-instance." The efficiency of a kernel (or kernelization algorithm) is determined by the size of the kernel. There are many parameterized problems that are fixed-parameter tractable but do not admit polynomial kernels unless NP $\subseteq$ coNP/poly. So, from the perspective of polynomial-time preprocessing, we look for kernels of polynomial-size.

*Graph Parameters:* In parameterized complexity, though the natural parameterization is the solution size, however, several structural graph parameters have also taken into account [11]. In our problem, the natural parameter is the vertex cover of the forcing graph. As mentioned in [3], the vertex cover can be computed in $\mathcal{O}^*(1.2738^k)$ time where $k$ is the size of the vertex cover. Another important graph parameter is the $\mathcal{G}$*-deletion set* where $\mathcal{G}$ is a graph class. A subset of the vertices $S \subseteq V(G)$ is said to be a *deletion set* to graph class $\mathcal{G}$ if $G - S \in \mathcal{G}$.

*Some Preliminary Algorithmic Results:* In this section, we establish some classical complexity dichotomy result and some related parameterized complexity results for this problem. The first part of this section gives a proof that the problem is polynomial-time solvable when the forcing graph is a $2K_2$-free graph. Towards this, we define the following annotated problem that would be useful for both the classical and parameterized complexity results.

Ext-SPFG

**Input:** A simple undirected graph $G(V, E)$, two distinct vertices $s, t \in V(G)$, a forcing graph $G_f(E, E^{'})$ and a vertex cover $S$ of $G_f$.

**Goal:** Find a subset $E^* \subseteq E$ with minimum number of edges such that $S \subseteq E^*$, i.e. $E^*$ extends $S$, and the induced subgraph in $G$ by the edge set $E^*$ contains an $s$-$t$ path in $G$.

Our next lemma provides a polynomial-time algorithm for Ext-SPFG.

**Lemma 1.** *($\star$)*[2] Ext-SPFG *can be solved in polynomial-time.*

Using the above lemma, we can provide a polynomial-time algorithm for SPFG the forcing graph is $2K_2$-free.

[2] Due to lack of space, the proofs that are omitted or marked $\star$ can be found in the full version (https://arxiv.org/abs/2309.04346).

**Lemma 2.** *($\star$) The* Shortest Path with Forcing Graph *can be solved in polynomial time if the forcing graph is $2K_2$-free.*

The above lemma illustrates that if the forcing graph is $2K_2$-free, then the optimization version of the Shortest Path with Forcing Graph can be solved in polynomial-time. Darman et al. [7] proved that Shortest Path with Forcing Graph is NP-Complete even when the forcing graph $G_f$ is a 2-ladder, i.e. graph of degree one. In particular, their construction ensures that there are several $2K_2$s present in the forcing graph as subgraphs. So, we complete this picture by the following dichotomy.

**Theorem 1.** Shortest Path with Forcing Graph *is polynomial-time solvable when the forcing graph is a $2K_2$-free graph and* NP-Complete *otherwise.*

After discussing the classical complexity of Shortest Path with Forcing Graph, we move on to discuss the parameterized complexity of the same. Since solution size is the most natural parameter; i.e. the number of edges in an optimal solution, we first prove that Shortest Path with Forcing Graph is FPT when parameterized by the solution size. For this purpose, we use the following existing result by Damaschke et al. [5].

**Proposition 2** [5]**.** *Given a graph $G$ and positive integer $k$, all the vertex cover of $G$ of size at most $k$ can be enumerated in $\mathcal{O}(m + 2^k k^2)$ time.*

We prove the following result by using Proposition 2 and Lemma 1.

**Theorem 2.** *($\star$) The* Shortest Path with Forcing Graph *is fixed-parameter tractable and can be solved in $\mathcal{O}((m + 2^k k^2)(m + n))$ time.*

## 3 Polynomial Kernels for SPFG

In the previous section, we have discussed that if Shortest Path with Forcing Graph is fixed-parameter tractable when there are no restrictions on the original graph $G$ and the forcing graph $G_f$, i.e., $G$ and $G_f$ are arbitrary graphs. This section is devoted to the kernalization complexity of Shortest Path with Forcing Graph problem when solution size is considered as the parameter. Our kernelization algorithm has intuitively two parts, "hitting the edges of $G_f$" and "providing connectivity between $s$ and $t$ in $G$". As the edges of $G$ are the vertices of $G_f$, we define the following edge subsets of $G$.

➢ We put an edge $e \in E(G)$ in $H$ if $deg_{G_f}(e) \geq k + 1$.
➢ We put $e \in E(G)$ in $L$ if $N_{G_f}(e) \subseteq H$.
➢ $R = E(G) \setminus (H \cup L)$.

Notice for any edge $e \in E(G)$, if $N_{G_f}(e) \subseteq H$, then $e \in L$. Hence, we have the following observation.

**Observation 1.** *If $e$ is an isolated vertex in $G_f$, then $e \in L$.*

Now, we prove the following lemma that will be one of the important parts in obtaining the kernel.

**Lemma 3.** *If $I(G, G_f, s, t, k)$ is a* Yes*-instance then $|H| \leq k$ and $G_f[R]$ has at most $k^2$ edges.*

*Proof.* By Observation 1, an isolated vertex $e$ of $G_f$ is in $L$. So, every $e \in R$ has at least one neighbor (with respect to $G_f$) in $R$. As any $e \in V(G_f)$ with degree at least $k+1$ in $G_f$ is put in the set $H$, any $e \in R$ must have at most $k$ neighbors in $R$. By our hypothesis, $I(G, G_f, s, t, k)$ is a Yes-instance. Hence, there is $E^* \subseteq E(G)$ such that $|E^*| \leq k$ for every $(a, b) \in E(G_f)$, at least one of $a$ and $b$ must be in $E^*$. If there is $e \in H \setminus E^*$, then at least $k+1$ edges have to be in $E^*$ that is a contradiction to the fact that $I(G, G_f, s, t, k)$ is a Yes-instance. Hence, $H \subseteq E^*$ implying that $|H| \leq k$. Consider the set $R$. As every $e \in R$ at least one neighbor belongs to $R$ in $G_f$ and at most $k$ neighbors belong to $R$ in $G_f$. Hence, the number of edges in $G_f$ that are incident to $R$ is at most $k^2$. Therefore the cardinality of $V(R)$ is at most $2k^2$. □

Observe that the edges in $H$ are necessary for any solution of size at most $k$ that certainly "hits every edge of $G_f$". But, the role of the edges in $L$ are only to provide connectivity between $s$ and $t$ in $G$. Let $E_k$ be the set of edges in $G_f[H \cup R]$. Recall, $V(E_k) = \{u, v | \ e(u, v) \in E_k\}$. For our convenience, we also add $s$ and $t$ into $V(E_k)$. More formally, $V(E_k) = V(E_k) \cup \{s, t\}$ and let $Y = V(G) \setminus V(E_k)$. We mark some additional vertices from $Y$ using the following marking scheme.

➢ For each pair $(x, y)$ of vertices in $V(E_k)$ compute a shortest $x$-$y$ path, $P_{xy}$ via the internal vertices of $Y$ in $G$.

➢ If $P_{x,y}$ has at most $k$ edges, then mark the edges of $P_{x,y}$.

➢ Else $P_{x,y}$ has more than $k$ edges. Then, do not mark any edge.

➢ Finally, for every pair $x, y \in V(E_k)$, mark the edges of a shortest path $Q_{x,y}$ in $G$ when $|Q_{x,y}| \leq k$.

Let $E_t = \bigcup\limits_{x,y \in V(E_k)} (P_{x,y} \cup Q_{x,y})$ be the set of marked edges of $G$ after the completion of the above marking scheme. Consider $E_M = E_t \cup H \cup R$. Formally, $E_M$ be the edges that are in $H \cup R$ as well as in $E_t$. We denote $G[E_M] = G(V, E_M)$ be the subgraph of $G$ induced by the set of edges in $E_M$ and consider the instance as $I(G[E_M], G_f[E_M], s, t, k)$. Next, we prove the following lemma.

**Lemma 4.** *The instance $I(G, G_f, s, t, k)$ is a* Yes*-instance if and only if $I(G[E_M], G_f[E_M], s, t, k)$ is a* Yes*-instance.*

*Proof.* Let us first give the backward direction ($\Leftarrow$) of the proof. First, assume that the instance $I(G[E_M], G_f[E_M], s, t, k)$ is a Yes-instance. One can make a note that edges present in $G[E_M]$ are also in $G$ and in $G_f[E_M]$ as vertices. Suppose that $G[E_M]$ contains a set of edges $E^*$ such that $G[E^*]$ has an $s$-$t$ path and $E^*$ is a vertex cover in $G_f$. Then, $H \subseteq E^*$ and hence $E^*$ also forms a vertex cover of $G_f$. Moreover, an $s$-$t$ path passing through a (proper) subset of edges in $E^*$ is also an $s$-$t$ path in $G$. Hence, $I(G, G_f, s, t, k)$ is a Yes-instance.

Next, we focus on proving the forward direction ($\Rightarrow$). Assume that $I(G, G_f, s, t, k)$ is a `Yes`-instance. Let $E^*$ be the solution to the instance $I(G, G_f, s, t, k)$ and let $P$ be an $s$-$t$ path contained inside the graph induced by $G(V, E^*)$. As $|E^*| \leq k$ is a solution to $I(G, G_f, s, t, k)$, $H \subseteq E^*$. If $E^* \subseteq E_M$, then $E^*$ is a solution to $I(G[E_M], G_f[E_M], s, t, k)$ and we are done. In case some edge $e \in E^* \setminus E_M$ does not belong to any $s$-$t$ path in $G(V, E^*)$, then clearly such an edge $e \in L$. We just replace that edge $e$ with $\hat{e}$ such that $\hat{e} \in N_{G_f}(e) \cap H$. Consider those edges that belong to some $s$-$t$ path in $G(V, E^*)$. Consider those subpaths (one at a time) $P^* \subseteq P$ that contains an edge $e \in E^* \setminus E_M$. Observe that $P^*$ has at most $k$ edges and is an $x$-$y$ path in $G$ for some $x, y \in V(E_k) \cup \{s, t\}$. But, we have marked a shortest path $\hat{P}^*$ from $x$ to $y$ in $G$ (either via the vertices of $Y$ or in $G$ itself). We just replace the edges of $P^*$ by $\hat{P}^*$. As $|\hat{P}^*| \leq |P^*|$, this constructs an $s$-$t$ walk. Similarly, for other subpaths also, we use the same replacement procedure and eventually construct an $s$-$t$ walk with at most $k$ edges in $G[E_M]$. As $\hat{E}^*$ provides an $s$-$t$ in $G[E_M]$, $\hat{E}^*$ is a solution to $I(G[E_M], G_f, s, t, k)$. □

Observe that for every pair of vertices in $V(E_k)$, we have marked a shortest path of length at most $k$ in $G$. We are ready to prove our final theorem statement.

**Theorem 3.** *The* Shortest Path with Forcing Graph *admits a kernel with $\mathcal{O}(k^5)$ vertices and edges.*

*Proof.* Our kernelization algorithm works as follows. First, we compute a partition of $V(G_f) = H \uplus R \uplus L$ as described. From Lemma 3, we have that $H \cup R$ has at most $\mathcal{O}(k^2)$ edges in $G_f$. After that, we invoke the marking scheme described. Observe that the marking scheme marks a shortest path of length at most $k$ for every pair of vertices $x, y \in V(E_k)$ and put them in $E_M$. Hence, $|E_M|$ is $\mathcal{O}(k^5)$. From Lemma 4, $I(G, G_f, s, t, k)$ is a `Yes`-instance if and only if $I(G[E_M], G_f[E_M], s, t, k)$ is a `Yes`-instance. Let $W = V(E_M)$, i.e. the vertices that are the endpoints of the edges of $E_M$ in $G$. We output $(G[W], G_f[E_M], s, t, k)$ as the output instance. As $|E_M|$ is $\mathcal{O}(k^5)$, $|W|$ is also $\mathcal{O}(k^5)$. Therefore, Shortest Path with Forcing Graph admits a kernel with $\mathcal{O}(k^5)$ vertices and edges. □

## 4 Improved Kernels for Special Graph Classes and Results on Structural Parameters

Consider an input instance $I(G, G_f, s, t, k)$ to Shortest Path with Forcing Graph. This section is devoted to kernelization algorithms when either $G$ or $G_f$ belongs to some special graph class. For both the results, we give a proof sketch here and refer to appendix for more detailed proofs.

**Theorem 4.** Shortest Path with Forcing Graph *admits a kernel with $\mathcal{O}(k^3)$ vertices when $G$ is a planar graph.*

*Proof.* (Sketch) The input graph $G$ has a special property satisfying Euler's formula but the forcing graph $G_f$ can be an arbitrary graph. If $G$ is a simple graph with $n$ vertices, then $G$ can have at most $3n - 6$ edges. We partition the vertices of $G_f$, i.e. the edges of $E(G)$ into $H, L$ and $R$ as inn the previous section. Put an edge $e \in E(G)$ into $H$ if $deg_{G_f}(e) > k$. Put $e \in E(G)$ into $L$ if $N_{G_f}(e) \subseteq H$. Define $R = E(G) \setminus (H \cup L)$. Our first step is to invoke Lemma 3, that if $I(G, G_f, s, t, k)$ is a `Yes`-instance, then $|H| \leq k$ and $G_f[R]$ has at most $k^2$ edges. Since every vertex of $G_f[H \cup R]$ is incident to some edge, $H \cup R$ has $\mathcal{O}(k^2)$ vertices that are edges of $G$. Let $V_L \subseteq V(G)$ denote the vertices spanned by the edges of $G$ present in $H \cup R$ and $V_I = V(G) \setminus V_L$. For every pair $\{x, y\}$ of $V_L$, we define a boolean variable $J_{(\{x,y\})}$ is true if there is a path from $x$ to $y$ in $G$ with internal vertices in $V_I$ and $J_{(\{x,y\})}$ is false otherwise. We prove a structural characterization which says that "there are $3|V_L| - 6$ distinct pairs of vertices $\{x, y\}$ in $V_L$ for which the boolean variable $J_{(\{x,y\})}$ is true". We now consider the a similar marking scheme $\mathsf{MarkPlanar}(G, G_f, s, t, k)$ as before. We give a description of this for the sake of completeness and clarity. For each pair $(x, y)$ of vertices from $V_L$, compute a shortest path $P_{x,y}$ from $x$ to $y$ that uses only the vertices of $V_I$ as internal vertices. If $P_{x,y}$ has at most $k$ edges, then mark the edges of $P_{x,y}$. Otherwise do not mark any edge of $P_{x,y}$. Finally, for every pair $(x, y)$ from $V_L$, mark the edges of a shortest path $Q_{x,y}$ from $x$ to $y$ in $G$ if $Q_{x,y}$ has at most $k$ edges. Consider $E_t \subseteq E(G)$, the set of all edges that are marked and let $E_M = E_t \cup H \cup R$. After that, we prove that "the instance $I(G, G_f, s, t, k)$ is equivalent to $I(G[E_M], G_f[E_M], s, t, k)$". Using this, we prove our result (Theorem 4) that SPFG admits a kernel with $\mathcal{O}(k^3)$ vertices when $G$ is a planar graph. □

**Theorem 5.** Shortest Path with Forcing Graph *admits a kernel with* $\mathcal{O}(k^3)$ *vertices when* $G_f$ *is either a cluster graph or a graph with bounded degree.*

*Proof.* (Sketch) The input graph $G$ is arbitrary here but the forcing graph $G_f$ can be either a cluster graph or a graph of bounded degree. In this situation, we exploit some special properties of cluster graphs or bounded degree graphs as follows. In particular, for the hitting part, we can ensure that $\mathcal{O}(k)$ vertices are sufficient for the hitting part, i.e. to hit all the edges of $G_f$. An intuition behind this is that if $C$ is a clique in a graph $G_f$, then at least $|C| - 1$ vertices of $C$ are part of any vertex cover of $G_f$. We first prove two statements. The first statement says "if $G_f$ is cluster graph and $I(G, G_f, s, t, k)$ is a `Yes`-instance, then $G_f$ has at most $2k$ vertices that are not isolated in $G_f$". The second statement says "$G_f$ is a bounded degree graph with maximum degree at most $\eta$ and $I(G, G_f, s, t, k)$ is a `Yes`-instance, then $G_f$ has at most $k\eta$ vertices that are not isolated in $G_f$". Let $V_L^f \subseteq V(G_f)$ be the set of vertices that are not isolated in $G_f$ and they are edges in $G$ and $V_L$ be the set of vertices of $G$ that are the endpoints of these edges of $V_L^f$. Consider $V_I = V(G) \setminus V_L$. We use a similar marking procedure as before. For each pair $(x, y)$ of vertices from $V_L$, compute a shortest path $P_{x,y}$ from $x$ to $y$ in $G$ that uses only the vertices of $V_I$ as internal vertices. If $P_{x,y}$ has at most $k$ edges, then mark the edges of $P_{x,y}$. Otherwise, when $P_{x,y}$ has more than $k$ edges, do not mark any edge of $P_{x,y}$. Finally, for each pair $(x, y)$ of vertices from $V_L$, compute a shortest path $Q_{x,y}$ from $x$ to $y$ in $G$ when $Q_{x,y}$ has at most $k$ edges. Let $E_t$ be the set of all the edges that are marked by the above mentioned marking scheme and let $E_M = E_t \cup V_L^f$. We consider $G[E_M]$ as the output instance and prove that "$I(G, G_f, s, t, k)$ is a `Yes`-instance if and only if the output instance $I(G[E_M], G_f[E_M], s, t, k)$ is a `Yes`-instance". Using this, we can prove (Theorem 5) that SPFG admits a kernel with $\mathcal{O}(k^3)$ vertices when $G_f$ is either a cluster graph or a graph of bounded degree. $\square$

*Results on Structural Parameterizations.* Now, we provide a short summary of our result the structural parameterization of Shortest Path with Forcing Graph. We primarily consider the case when the deletion distance ($k$) to $2K_2$-free graph of $G_f$. Our first step here is prove the following lemma that enumerates all minimal vertex covers of a $2K_2$-free graph.

**Lemma 5.** *($\star$) Given an instance $(G, G_f, X, s, t, \ell)$ to the* SPFG-$2K_2$-Free-Deletion *problem, the set of all minimal vertex covers of $G$ can be enumerated in $2^{|X|}n^{\mathcal{O}(1)}$-time.*

Using the above lemma, we can prove the following theorem saying that SPFG-$2K_2$-Free-Deletion is fixed-parameter tractable.

**Theorem 6.** *($\star$)* SPFG-$2K_2$ -Free-Deletion *admits an algorithm that runs in $2^{|X|}n^{\mathcal{O}(1)}$-time.*

## 5 Conclusion and Open Problems

In this paper, we have initiated the study of Shortest Path with Forcing Graph under the realm of parameterized complexity. One natural open problem is to see if our kernelization results for Shortest Path with Forcing Graph can be improved, i.e. can we get a kernel with $\mathcal{O}(k^4)$ vertices for SPFG when both $G$ and $G_f$ are arbitrary graphs? We strongly believe that those results can be improved but some other nontrivial techniques might be necessary. In. addition, it would be useful to have a systematic study of this problem under positive disjunctive constraints containing three (or some constant number of) variables. From the perspective of kernelization complexity, we leave the following open problems for future research directions.

➢ Can we get a kernel with $\mathcal{O}(k^4)$ vertices for SPFG when the input graph $G$ is arbitrary graph but the forcing graph $G_f$ is a graph of degeneracy $\eta$? Our results only show that if the forcing graph is of bounded degree, then we can get a kernel with $\mathcal{O}(k^3)$ vertices. In fact, even if $G_f$ is a forest, then also it is unclear if we can get a kernel with $\mathcal{O}(k^3)$ or $\mathcal{O}(k^4)$ vertices.

➢ What happens to the kernelization complexity when $G_f$ is an interval graph while $G$ is an arbitrary graph? Can we get a kernel with $\mathcal{O}(k^3)$ vertices in such case?

➢ Finally, can we generalize our result of Theorem 4 when $G$ is a graph of bounded treewidth or graph of bounded degeneracy?

## References

1. Agrawal, A., Jain, P., Kanesh, L., Saurabh, S.: Parameterized complexity of conflict-free matchings and paths. In: Algorithmica, pp. 1–27 (2020)
2. Cai, L., Chen, J., Downey, R.G., Fellows, M.R.: Advice classes of parameterized tractability. Ann. Pure Appl. Logic **84**(1), 119–138 (1997)
3. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. **411**(40–42), 3736–3756 (2010)
4. Cygan, M., et al.: Parameterized Algorithms, vol. 5. Springer, Heidelberg (2015)
5. Damaschke, P.: Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. Theor. Comput. Sci. **351**(3), 337–350 (2006)
6. Darmann, A., Pferschy, U., Schauer, J.: Determining a minimum spanning tree with disjunctive constraints. In: Rossi, F., Tsoukias, A. (eds.) ADT 2009. LNCS (LNAI), vol. 5783, pp. 414–423. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04428-1_36
7. Darmann, A., Pferschy, U., Schauer, J., Woeginger, G.J.: Paths, trees and matchings under disjunctive constraints. Disc. Appl. Math. **159**(16), 1726–1735 (2011)
8. Diestel, R.: Graph theory 3rd ed. Graduate texts in mathematics, vol. 173 (2005)
9. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (2012)
10. Even, G., Halldórsson, M.M., Kaplan, L., Ron, D.: Scheduling with conflicts: online and offline algorithms. J. Sched. **12**(2), 199–224 (2009)
11. Guo, J., Hüffner, F., Niedermeier, R.: A structural view on parameterizing problems: distance from triviality. In: Downey, R., Fellows, M., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 162–173. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28639-4_15
12. Jain, P., Kanesh, L., Misra, P.: Conflict free version of covering problems on graphs: classical and parameterized. Theory Comput. Syst. **64**(6), 1067–1093 (2020)
13. Majumdar, D., Ramanujan, M.S., Saurabh, S.: On the approximate compressibility of connected vertex cover. Algorithmica **82**(10), 2902–2926 (2020)
14. Pferschy, U., Schauer, J.: The maximum flow problem with disjunctive constraints. J. Comb. Optim. **26**(1), 109–119 (2013)
15. West, D.B.: Introduction to Graph Theory. Pearson Education, Boston (2007)

# An Approximation Algorithm for the (Metric) Clustered Path Traveling Salesman Problem

Jiaxuan Zhang[1], Suogang Gao[1], Bo Hou[1,2], and Wen Liu[1,2](✉)

[1] School of Mathematical Sciences, Hebei Normal University, Shijiazhuang 050024, People's Republic of China
[2] Hebei Key Laboratory of Computational Mathematics and Applications, Shijiazhuang 050024, People's Republic of China
liuwen1975@126.com

**Abstract.** We consider the (metric) clustered path traveling salesman problem. In this problem, we are given a complete graph $G = (V, E)$ along with a nonnegative edge cost function satisfying the triangle inequality, where $V$ is partitioned into disjoint subsets $V_1, \ldots, V_k$ called clusters and $s \in V_1, t \in V_k$ are two given vertices. The objective of the problem is to find a Hamiltonian path of $G$ with minimum cost from $s$ to $t$, satisfying that all vertices in each cluster are visited consecutively by this path. In this paper, we consider the problem in the case where endpoints of the subpath induced by the path on each cluster are both specified and a 2-approximation algorithm is given.

**Keywords:** Approximation algorithm · Traveling salesman problem · Rural postman problem · Path · Cluster

## 1 Introduction

The (metric) traveling salesman problem (TSP), as a classical optimization problem, arises canonically in many applied settings and a vast number of its variations are studied both from a theoretical and an applied perspective [4,5,7,14,18,20,22] . In this problem, we are given a complete graph $G = (V, E)$ with edge cost $w(e) \in \mathbb{R}_{\geq 0}$ for each $e \in E$ satisfying triangle inequality. The task is to find a minimum-cost Hamiltonian cycle in $G$. The TSP is a typical example of an NP-hard problem [11]. Heuristic algorithms and approximation algorithms are given and the most-well known algorithm is the 1.5-approximation algorithm independently designed by Christofides [7] and Serdjukov [23].

The path traveling salesman problem (PTSP) is an important variant of the TSP. Instead of a cycle, the task of the PTSP is to find a minimum-cost Hamiltonian path. In the early 1990s, Hoogeveen [16] presented a $\frac{5}{3}$-approximation algorithm. An et al. [1] improved this result to $\frac{1+\sqrt{5}}{2}$. After successive improvements [13,21,25], Zenklusen [27] developed the best known $\frac{3}{2}$-approximation algorithm. Sun et al. [24] designed

https://doi.org/10.1007/978-981-96-4448-3_3

a $\frac{1+\sqrt{5}}{2}$-approximation algorithm for the problem by combining linear programming rounding strategy and a special structure for a more generalized path traveling salesman problem.

Another important variant of the TSP called the clustered traveling salesman problem (CTSP) is posed when we consider distinct priorities of different vertices. In the CTSP, in addition to the input for the TSP, the vertex set $V$ is partitioned into nonempty disjoint subsets $V_1, \ldots, V_k$ called clusters. The task is to find a minimum Hamiltonian cycle in $G$, where all vertices of each cluster are visited consecutively. Note that if $k = 1$ or $k = |V|$, the CTSP is exactly the TSP. So in the following we assume $1 < k < |V|$. Chisman [6] first introduced the CTSP and described some applications about it. Jongens and Volgenant [17] described an extensive adaptation of an existing traveling salesman algorithm based on the 1-tree relaxation. Gendreau et al. [12] propose new heuristics for the TSP with backhauls. Arkin et al. [3] developed a $\frac{7}{2}$-approximation algorithm for the CTSP. Anily et al. [2] considered the ordered cluster traveling salesman problem and the ordered cluster traveling salesman path problem, and developed a $\frac{5}{3}$-approximation algorithm. Guttmann-Beck et al. [15] designed approximation algorithms for several cases of the CTSP by decomposing them into the PTSP together with the stacker crane problem, or the PTSP with the rural postman problem. Kawasaki and Takazawa [19] designed improved approximation algorithms by applying Zenklusen's recent result for the PTSP in [27].

The clustered path traveling salesman problem (CPTSP) is a combination of both the PTSP and the CTSP. In the CPTSP, besides the input of the CTSP, we are given two endpoints $s \in V_1, t \in V_k$. The task is to find a minimum Hamiltonian path of $G$ from $s$ to $t$, where all vertices in each cluster are visited consecutively by this path. As mentioned above, $s$ and $t$ are the start vertex and the end vertex of the Hamiltonian path respectively, which implies that $V_1$ is the first and $V_k$ is the last cluster to be visited. For the remaining $k-2$ clusters, we visit them in any order. Note that a Hamiltonian path of the graph $G$ induces a Hamiltonian subpath on each cluster and each induced subpath also has its start vertex and end vertex.

For the CPTSP where the endpoints of the induced subpath on each cluster are both specified, and moreover, even the start vertex and the end vertex of each induced subpath are both specified, we designed an approximation algorithm by decomposing it into the PTSP and the path version the stacker crane problem [28]. In this paper, we consider the CPTSP where the endpoints of the induced subpath on each cluster are both specified, but it is free to choose either of them as the start vertex except for $V_1$ and $V_k$. By referencing the work of Anily et al. [2], Frederickson et al. [10], and Kawasaki and Takazawa [19], we design an approximation algorithm with an approximation ratio 2 for the CPTSP by decomposing it into the PTSP and the path version of the rural postman problem.

This remainder of this paper is organized as follows. In Sect. 2, we present some basic terminology, notation and results. In Sect. 3, we design two approximation algorithms for the PRPP. In Sect. 4, by combining these two algorithms given in Sect. 3, we design an algorithm for the CPTSP. Besides, we analyze its approximation ratio. In Sect. 5, we provide several problems for future research.

## 2 Preliminaries

In this section, we present some basic terminology, notation and results.

A *graph* $G$ is a pair $(V,E)$, where $V$ is a set of objects called *vertices* and $E$ is a set of *edges*. Each edge is a pair $\{u,v\}$ of vertices $u,v \in V$. Multiple edges connected to the same pair of vertices are called *parallel edges*. A graph is called a *multigraph* if it has parallel edges.

Let $S$ be a subset of $E$. If no two edges in $S$ have common vertices, then we say that $S$ is a *matching* of $G$. A matching $S$ of $G$ is called a *perfect matching* if every vertex of $G$ is an endpoint of some edge in $S$.

A *walk* in a graph $G$ from vertex $v_1$ to $v_l$ is a sequence $v_1, \{v_1,v_2\}, v_2, \{v_2,v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1},v_l\}, v_l$, in which all $v_i$s are vertices and $\{v_{i-1},v_i\} \in E$ for $i = 2,3,\ldots,l$. $v_1$ is called the *start vertex* and $v_l$ the *end vertex* and they are both called *endpoints* of the walk. A *path* is a walk with no repeated vertices. A *trail* is a walk with no repeated edges. An *Eulerian trail* is a walk passing through every edge of $G$ exactly once. If an Eulerian trail is closed (the start vertex is the same as the end vertex), it is called an *Eulerian tour*. A *Hamiltonian path/cycle* of $G$ is a path/cycle visiting every vertex of $G$ exactly once.

The Eulerian trail has the following important property.

**Lemma 1** [8]**.** *A connected (multi)graph has an Eulerian trail if and only if it has either 0 or 2 vertices of odd degree.*

In the following we present the rural postman problem and the path version of the rural postman problem. For any positive integer $k$, we use the notation $[k]$ to denote the set $\{1,2,\ldots,k\}$. In the rural postman problem, we are given a multigraph $G' = (V', E' \cup D)$, where $V' = \{s_i, t_i \mid i \in [k]\}$, $(V', E')$ is an undirected complete graph with a nonnegative edge cost function $w$ satisfying the triangle inequality and $D = \{\{s_i,t_i\} \mid i \in [k]\}$. The objective is to find a minimum-cost Hamiltonian cycle that traverses all edges in $D$.

If the objective is to find a minimum-cost Hamiltonian path from $s_1$ to $t_k$ that traverses all edges in $D$, we obtain the path version of the rural postman problem, and we call it the path rural postman problem (PRPP).

## 3 Approximation Algorithms for the Path Rural Postman Problem

In this section, we present two polynomial-time algorithms for the PRPP which will be used when we design the algorithm for the CPTSP in next section.

Recall that in the PRPP, we are given a multigraph $G' = (V', E' \cup D)$, where $V' = \{s_i, t_i \mid i \in [k]\}$, $(V', E')$ is an undirected complete graph with an edge cost function $w$ satisfying the triangle inequality, and $D = \{\{s_i,t_i\} \mid i \in [k]\}$. The objective is to find a minimum-cost Hamiltonian path from $s_1$ to $t_k$ that traverses all edges in $D$.

The first algorithm for the PRPP is as follows.

**Algorithm 1**

**Input:** A multigraph $G' = (V', E' \cup D)$, where $V' = \{s_i, t_i \mid i \in [k]\}$, $(V', E')$ is an undirected complete graph with a nonnegative edge cost function $w$ satisfying the triangle inequality, and $D = \{\{s_i, t_i\} \mid i \in [k]\}$.
**Output:** Path $P_{\text{PRPP1}}$.
**Step 1:** Except for $s_1, t_k$, identify vertices in the graph $G' = (V', E' \cup D)$, and perform a minimum perfect matching for them.
**Step 2:** Initialize $E_1$ to be empty. Insert all edges in the above matching into $E_1$. (This results in $m \geq 1$ disjoint connected components on $E_1 \cup D$, and we denote these $m$ connected components by $R_i$, $i \in [m]$.)
**Step 3:** Condense each $R_i$ into a single vertex $n_i$. Define

$$d(n_i, n_j) = \min\{w\{u, v\} \mid u \in R_i{}', v \in R_j{}'\},$$

where $R_i{}'$ represents the set of all vertices in $R_i$, except for $s_1, t_k$.
**Step 4:** Find a minimum spanning tree for the vertices in $\{n_i \mid i \in [m]\}$, where the minimum is with respect to the distance function $d$ defined at Step 3.
**Step 5:** Make two copies of each edge in the spanning tree, and insert these copied edges into $E_1$. (This results in a graph $G_1' = (V', E_1 \cup D)$.)
**Step 6:** Find an Eulerian trail from $s_1$ to $t_k$ in $G_1'$.
**Step 7:** Shortcut vertices visited more than once to get a Hamiltonian path $P_{\text{PRPP1}}$ from $s_1$ to $t_k$ traversing all edges in $D$.

Assume $\text{OPT}'$ is the cost of an optimal solution of the PRPP and $U = \sum_{i=1}^{k} w\{s_i, t_i\}$.

**Lemma 2.** *Algorithm 1 outputs a Hamiltonian path $P_{PRPP1}$ with cost at most* $3\text{OPT}' - 2\text{U}$.

*Proof.* We first show that $P_{\text{PRPP1}}$ is a Hamiltonian path from $s_1$ to $t_k$ that traverses all edges in $D$. According to Step 2, we know that, except for $s_1, t_k$, the degree of each vertex is even in these connected components. The edges of the spanning tree created at Step 4 connect these disjoint connected components produced at Step 2 into one. Since at Step 5 for each edge of the spanning tree, we make two copies, the degree of each vertex is still even in the one connected component, except for $s_1, t_k$. Step 1 and Step 3 of the algorithm guarantee that the degrees of $s_1$ and $t_k$ are both 1. At Step 6, by Lemma 1, we get the Eulerian trail from $s_1$ to $t_k$ in $G_1'$. At Step 7, using the triangle inequality, we get a desired Hamiltonian path.

In the following, we consider the cost of $P_{\text{PRPP1}}$. Since an optimal path of this problem contains all edges in $D$ and a perfect matching for all vertices, except for $s_1, t_k$, its cost can not be smaller than that of the minimum perfect matching obtained at Step 1 and the edges in $D$. For convenience, we denote the cost of the minimum perfect matching by $M$. Then $M + U \leq \text{OPT}'$. Since the edges in the optimal path, except for the edges in $D$, can connect these disjoint connected components into one and the spanning tree created at Step 4 also connects these disjoint connected components into one, the cost of the minimum spanning tree at Step 4 must be no greater than $\text{OPT}' - U$. Therefore, the cost of the Eulerian trail at Step 6 is at most $M + U + 2(\text{OPT}' - U)$, and then is at most $3\text{OPT}' - 2U$. Using the triangle inequality, we get the cost of the path $P_{\text{PRPP1}}$ is at most the cost of the Eulerian trail. □

The following is the second algorithm for the PRPP.

**Algorithm 2**

```
Input: A multigraph $G' = (V', E' \cup D)$, where $V' = \{s_i, t_i \mid i \in [k]\}$, $(V', E')$ is an undirected complete graph with a nonnegative edge cost function $w$ satisfying the triangle inequality, and $D = \{\{s_i, t_i\} \mid i \in [k]\}$.
Output: Path $P_{\mathrm{PRPP2}}$.
Step 1: Condense each edge in $D$ into a vertex $n_i$. For each pair $i, j$ with $i, j \in [k]$, define
    $d(n_i,n_j) = \begin{cases} \min\{w\{s_i,s_j\}, w\{s_i,t_j\}, w\{t_i,s_j\}, w\{t_i,t_j\}\}, & \text{if } s_1, t_k \notin \{s_i,s_j,t_i,t_j\}, \\ \min\{w\{t_i,s_j\}, w\{t_i,t_j\}\}, & \text{if } s_1 = s_i, \\ \min\{w\{s_i,t_j\}, w\{t_i,t_j\}\}, & \text{if } s_1 = s_j, \\ \min\{w\{s_i,t_j\}, w\{s_i,s_j\}\}, & \text{if } t_k = t_i, \\ \min\{w\{t_i,s_j\}, w\{s_i,s_j\}\}, & \text{if } t_k = t_j, \\ w\{t_i,s_j\}, & \text{if } s_1 = s_i, t_k = t_j, \\ w\{s_i,t_j\}, & \text{if } s_1 = s_j, t_k = t_i. \end{cases}$
    .
Step 2: Find a minimum spanning tree for the vertices in $\{n_i \mid i \in [k]\}$, where the minimum is with respect to the distance function $d$ defined at Step 1.
Step 3: Initialize $E_2$ to be empty. Insert all edges in the above spanning tree into $E_2$. Restore the vertex $n_i$ with the corresponding edge $\{s_i, t_i\}$, $i \in [k]$. (This results in a graph with vertex set $V'$ and edge set $E_2 \cup D$.)
Step 4: Except for $s_1, t_k$, identify vertices of odd degree in the graph obtained at Step 3, and find a minimum perfect matching for them.
Step 5: Insert all edges in the above matching into $E_2$. (This results in a graph $G'_2 = (V', E_2 \cup D)$. Note that this graph $(V', E_2 \cup D)$ got in this step is different with that in Step 3, because the set $E_2$ is updated.)
Step 6: Find an Eulerian trail from $s_1$ to $t_k$ in $G'_2$.
Step 7: Shortcut vertices visited more than once to get a Hamiltonian path $P_{\mathrm{PRPP2}}$ from $s_1$ to $t_k$ traversing all edges in $D$.
```

**Lemma 3.** *Algorithm 2 outputs a Hamiltonian path $P_{PRPP2}$ with cost at most* $2\mathrm{OPT}'$.

*Proof.* With a similar discussion with the proof of Lemma 2, after completion of Step 5, all vertices are of even degree, except for $s_1, t_k$. The degrees of $s_1$ and $t_k$ are both 1. This is obtained by the definition of $d(,)$ at Step 1 and by ignoring these two odd degree vertices $s_1, t_k$ at Step 4. According to Lemma 1, we find an Eulerian trail from $s_1$ to $t_k$ in $G'_2$. At Step 7, using the triangle inequality, we get a desired Hamiltonian path.

In the following, we consider the cost of $P_{\mathrm{PRPP2}}$. Since the edges in the optimal path, except for the edges in $D$, can connect the vertices in $\{n_i \mid i \in [k]\}$, the cost of the spanning tree at Step 2 is at most $\mathrm{OPT}' - U$. Then the cost of the graph obtained at Step 3 is at most $\mathrm{OPT}'$. So the cost of the minimum perfect matching on these vertices of odd degree, except for $s_1, t_k$, is at most $\mathrm{OPT}'$. Therefore the cost of the graph obtained at Step 5 is at most $2\mathrm{OPT}'$, and so the cost of the Eulerian trail at Step 6 is at most $2\mathrm{OPT}'$.

Also using the triangle inequality, we get the cost of the Hamiltonian path $P_{\mathrm{PRPP2}}$ is at most $2\mathrm{OPT}'$, as desired. □

*Remark 1.* Each of Algorithm 1 and Algorithm 2 is run, and the value $U$ relative to $\mathrm{OPT}'$ will determine which algorithm we will choose. For specific, if $U \geq \frac{1}{2}\mathrm{OPT}'$, we choose Algorithm 1 to run. Choose the other one instead.

**Lemma 4.** *The time complexity of Algorithm 1 and Algorithm 2 is*

$$O(\max\{\left|V'\right|^2\left|E'\right|, \left|E'\right|\log\log\left|V'\right|\}),$$

*where $V'$ and $E'$ are the set of vertices and edge subset given in the PRPP.*

*Proof.* The running time of Algorithm 1 and Algorithm 2 depends on the running time of finding the minimum perfect matching and the minimum spanning tree. There is a Blossom exact algorithm for computing minimum perfect matching by Edmonds [9]. The running time of the Blossom algorithm for finding the minimum perfect matching is $O(|V'|^2\,|E'|)$. According to Yao [26], the running time of the minimum spanning tree exact algorithm is $O(|E'|\log\log|V'|)$. □

## 4 An Approximation Algorithm for the Clustered Path Traveling Salesman Problem

In this section, we first design an approximation algorithm for the CPTSP, then we analyze its approximation ratio.

Recall that in the CPTSP, the endpoints of cluster $V_i$ for each $i \in [k]$ are both specified, but we are free to choose either of them as the start vertex except for the clusters $V_1, V_k$. In order to apply the algorithms designed in Sect. 2 for the PRPP, we assume that the endpoints are $s_i, t_i$ for each cluster $V_i$, $i \in [k-1]\backslash\{1\}$, the start vertices are $s_1, s_k$ and the end vertices are $t_1, t_k$ for $V_1, V_k$, respectively. Obviously, $s_1$ coincides with $s$ and $t_k$ coincides with $t$. Our algorithm mainly consists of four steps. At Step 1, for each fixed $i \in [k]$, we find the $path_i$, a path between $s_i$ and $t_i$ that goes through all vertices in $V_i$. This is exactly the PTSP with given endpoints. At Step 2, we replace the $path_i$ by the corresponding edge $\{s_i, t_i\}$, $i \in [k]$. At Step 3, we apply Algorithm 1 and Algorithm 2 to find a Hamiltonian path $P_{\mathrm{PRPP}}$ from $s_1$ to $t_k$ traversing each edge in $D$. At Step 4, we replace the edges in $D$ by the path $path_i$ ($i \in [k]$) obtained at Step 1.

**Algorithm 3**

**Input:** A complete graph $G = (V, E)$ with an edge cost function $w: E \longrightarrow \mathbb{R}_{\geq 0}$, clusters $V_1, \ldots, V_k$, endpoints $s_i$ and $t_i$, for each $V_i$, $i \in [k]$, start vertex $s_1 \in V_1$, and end vertex $t_k \in V_k$.
**Output:** Hamiltonian path $T$.
**Step 1:** For each $V_i$, $i \in [k]$, compute $path_i$, a Hamiltonian path with endpoints $s_i$ and $t_i$.
**Step 2:** Replace the $path_i$ by the corresponding edge $\{s_i, t_i\}$, $i \in [k]$. (This results in a multigraph $(V', E' \cup D)$ with vertex set $V' = \{s_i, t_i \mid i \in [k]\}$, $V'$ and edge subset $E'$ form a complete graph and $D = \{\{s_i, t_i\} \mid i \in [k]\}$.)
**Step 3:** Apply Algorithm 1 and Algorithm 2 to find the solution $P_{\mathrm{PRPP}}$ of PRPP with less cost in the multigraph obtained above.
**Step 4:** In $P_{\mathrm{PRPP}}$, replace the edges in $D$ by $path_i$, $i \in [k]$, we get a Hamiltonian path $T$ from $s_1$ to $t_k$.

*Remark 2.* Although the start vertices and the end vertices of $path_1$,$path_k$ are both determined, and we should replace them by the corresponding directed edges at Step 2, we can ensure that the directed edges are correctly traversed by finding the solution of PRPP because of the special positions of these two directed edges. Hence, Algorithm 1 and Algorithm 2 are simplified by replacing these two directed edges with undirected ones, and at Step 4, this is also true when replace the edges with the corresponding paths.

In this algorithm, we are involved in the PTSP and the PRPP that are both solvable in polynomial time, so our algorithm runs in polynomial time.

*Example 1.* A sample execution of the algorithm.

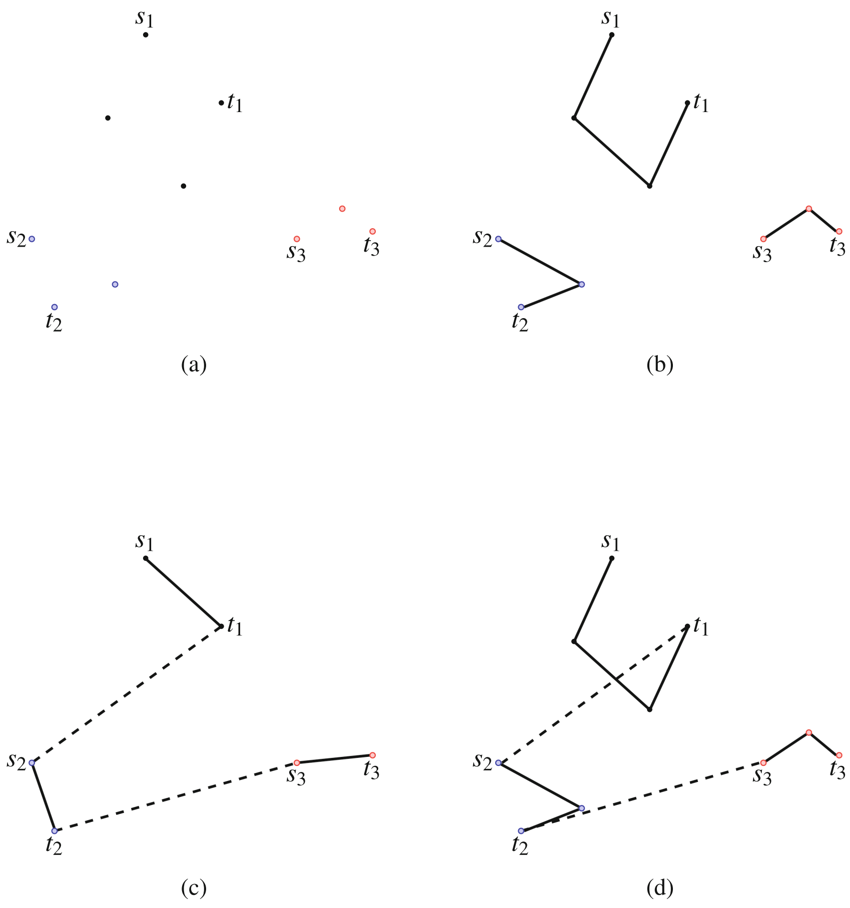


**Fig. 1.** Illustration of Algorithm 3

In this example, we are given three clusters (see Fig. 1(a)). We first compute $path_i$ with specified endpoints in each cluster $V_i$, $i \in [3]$ (see Fig. 1(b)). Then we replace the $path_i$

with the edge $\{s_i, t_i\}$ and solve the PRPP instance (see Fig. 1(c)). At last we replace the edge $\{s_i, t_i\}$ by the $path_i$ in $V_i$, $i \in [3]$ (see Fig. 1(d)).

Let OPT$''$ denote the cost of an optimal solution of the PTSP. The following result for the PTSP is useful for the analysis of Algorithm 3.

**Lemma 5** [19]. *For the PTSP with $u_1$ and $u_2$ being the two given endpoints, there exists a polynomial-time approximation algorithm that finds Hamiltonian paths $S_1$ and $S_2$ between $u_1$ and $u_2$ such that $w(S_1) \leq 2\mathrm{OPT}'' - w\{u_1, u_2\}, w(S_2) \leq \frac{3}{2}\mathrm{OPT}''$.*

*Remark 3.* We record the Hamiltonian path with lower cost as the output path of the PTSP algorithm.

Let OPT denote both an optimal solution of the CPTSP and its total cost.

**Theorem 1.** *Let $T$ be the path output by Algorithm 3. Then $w(T) \leq 2\mathrm{OPT}$.*

*Proof.* The algorithm consists of two subproblems: the PTSP and the PRPP. Let $P_i$ denote the induced path of OPT on $V_i$, $i \in [k]$. Let $L = \sum_{i\in[k]} \sum_{e\in P_i \cap \mathrm{OPT}} w(e)$, $L' = \mathrm{OPT} - L$. Recall that $U = \sum_{i=1}^{k} w\{s_i, t_i\}$. By Lemma 5, we get $\sum_{i=1}^{k} w(path_i) \leq \min(2L - U, \frac{3}{2}L)$. Therefore, $\sum_{i=1}^{k} w(path_i) \leq \min(2L - U, \frac{3}{2}L) \leq 2L - U$.

Note that there exists a solution of PRPP with cost $L' + U$. Hence, by Lemmas 2 and 3, the costs of the two solutions returned by the PRPP algorithm are at most $3L' + U$ and $2L' + 2U$, respectively. Therefore we get

$$\begin{aligned} w(P_{\mathrm{PRPP}}) &\leq \min(3L' + U, 2L' + 2U) \\ &\leq 2L' + 2U. \end{aligned}$$

At Step 4 of Algorithm 3, replace the edge $\{s_i, t_i\}$ in the solution of PRPP by $path_i$ for each $i \in [k]$. Then we obtain an upper bound on the cost of the solution $T$:

$$\begin{aligned} w(T) &= \sum_{i=1}^{k} w(path_i) - U + w(P_{\mathrm{PRPP}}) \\ &\leq (2L - U) - U + 2L' + 2U \\ &= 2L + 2L' = 2\mathrm{OPT}, \end{aligned}$$

as desired. □

## 5 Discussions

In this paper, we first design two approximation algorithms for the path rural postman problem, then based on them we design a 2-approximation algorithm for the clustered path traveling salesman problem in the case where the endpoints are specified in each cluster but we are free to choose the start vertex except for $V_1, V_k$. Other cases including only the start vertex is given in each cluster, and neither of the endpoints is given in each cluster, are also worth considering. Besides, to consider whether there exists any inapproximability result for the PRPP and the CPTSP is also challenging and interesting.

**Acknowledgements.** This work was supported by the NSF of China (No.11971146), the NSF of Hebei Province of China (No. A2023205009), the Local Science and Technology Development Funds Guided by the Central Government (No. 246Z7605G), and the Interdisciplinary Research Foundation of Hebei Normal University (No. L2024J01).

**Disclosure Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. An, H.C., Kleinberg, R., Shmoys, D.B.: Improving Christofides' algorithm for the s-t path TSP. J. ACM **62**(5), 1–28 (2015)
2. Anily, S., Bramel, J., Hertz, A.: A 5/3-approximation algorithm for the clustered traveling salesman tour and path problems. Oper. Res. Lett. **24**, 29–35 (1999)
3. Arkin, E.M., Hassin, R., Klein, L.: Restricted delivery problems on a network. Networks **29**, 205–216 (1994)
4. Bland, R.G., Shallcross, D.F.: Large travelling salesman problems arising from experiments in X-ray crystallography: a preliminary report on computation. Oper. Res. Lett. **8**(3), 125–128 (1989)
5. Blauth, J., Nägele, M.: An improved approximation guarantee for prize-collecting TSP. In: Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC), pp. 1848–1861. ACM, New York (2023)
6. Chisman, J.A.: The clustered traveling salesman problem. Comput. Oper. Res. **2**, 115–119 (1975)
7. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University (1976)
8. Diestel, R.: Graph Theory. Springer, New York (2017)
9. Edmonds, J.: Paths, trees and flowers. Can. J. Math. **17**, 449–467 (1965)
10. Frederickson, G.N., Hecht, M.S., Kim, C.E.: Approximation algorithms for some routing problems. SIAM J. Comput. **7**(2), 178–193 (1978)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco (1979)
12. Gendreau, M., Hertz, A., Laporte, G.: The traveling salesman problem with backhauls. Comput. Oper. Res. **23**, 501–508 (1996)
13. Gottschalk, C., Vygen, J.: Better s-t-tours by Gao trees. Math. Program. **172**, 191–207 (2018)
14. Grötschel, M., Holland, O.: Solution of large-scale symmetric traveling salesman problems. Math. Program. **51**, 141–202 (1991)
15. Guttmann-Beck, N., Hassin, R., Khuller, S., Raghavachari, B.: Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. Algorithmica **28**, 422–437 (2000)
16. Hoogeveen, J.A.: Analysis of Christofides' heuristic: some paths are more difficult than cycles. Oper. Res. Lett. **10**, 291–295 (1991)
17. Jongens, K., Volgenant, T.: The symmetric clustered traveling salesman problem. Eur. J. Oper. Res. **19**, 68–75 (1985)
18. Karlin, A.R., Klein, N., Gharan, S.O.: An improved approximation algorithm for TSP in the half integral case. In: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC), pp. 28–39. ACM, New York (2019)

19. Kawasaki, M., Takazawa, T.: Improving approximation ratios for the clustered travelling salesman problem. J. Oper. Res. Soc. Jpn. **63**(2), 60–70 (2020)
20. Plante, R.D., Lowe, T.J., Chandrasekaran, R.: The product matrix travelling salesman problem: an application and solution heuristics. Oper. Res. **35**, 772–783 (1987)
21. Sebő, A., Zuylen, A.V.: The salesman's improved paths: A 3/2+1/34 approximation. In: Proceedings of 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 118–127. IEEE, New Brunswick (2016)
22. Shmoys, D.B., Williamson, D.P.: Analyzing the Held-Karp TSP bound: a monotonicity property with application. Inf. Process. Lett. **35**(6), 281–285 (1990)
23. Serdjukov, A.: Some extremal bypasses in graphs. Upr Sist **17**, 76–79 (1978)
24. Sun, J., Gutin, G., Li, P., Shi, P., Zhang, X.: An LP-based approximation algorithm for the generalized traveling salesman path problem. Theor. Comput. Sci. **941**, 180–190 (2023)
25. Traub, V., Vygen, J.: Approaching $3/2$ for the *s*-*t* path TSP. J. ACM **66**(2), 1–17 (2019)
26. Yao, A.: An $O(|E|\log\log|V|)$ algorithm for finding minimum spanning trees. Inf. Process. Lett. **4**, 21–23 (1975)
27. Zenklusen, R.: A 1.5-approximation for path TSP. In: Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1539–1549. SIAM, San Diego (2019)
28. Zhang, J.X., Gao, S.G., Hou, B., Liu, W.: An approximation algorithm for the clustered path traveling salesman problem. J. Comb. Optim. **45**, 104 (2023)

# Dynamic Algorithms for Submodular Maximization with a $p$-Matchoid Constraint

Luying Ma, Yuanyuan Qiang, and Bin Liu(✉)

School of Mathematical Sciences, Ocean University of China, Qingdao, China
binliu@ouc.edu.cn

**Abstract.** Submodular maximization has been increasingly used in multiple applications of machine learning and data mining. Moreover, there has been a growing interest in both submodular maximization and dynamic algorithms. Previous work has considered dynamic algorithms for submodular functions under the cardinality constraint and the matroid constraint. In this paper, we consider the problem of maximizing a monotone submodular set function $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}^+$ subject to the $p$-matchoid constraint in the fully dynamic setting, where elements can be both inserted and deleted in real-time. Our main result is a randomized algorithm that obtains a $4p$-approximate solution and maintains an efficient data structure with a $\tilde{O}(k^2)$ amortized update time, where $k$ is an upper bound on the cardinality of the feasible solution.

**Keywords:** Submodular maximization · Dynamic algorithms · $p$-matchoid constraint

## 1 Introduction

Submodular functions play a fundamental role in classical combinatorial optimization. Many theoretical problems are instances of submodular functions, such as rank functions of matroids, edge cuts and coverage [15,25]. More recently, there is a large interest in constrained submodular function optimization driven both by theoretical progress and a variety of applications in computer science. Some of these include data summarization [12,21,27], influence maximization in social networks [9,10,16,18,26], generalized assignment [6], mechanism design [1], and network monitoring [20].

Given a collection $\mathcal{N}$ of items, also called the *ground set*, let $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}$ be a set function on a ground set $\mathcal{N}$. $f$ is *non-negative* if $f(S) \geq 0$ for any $S \subseteq \mathcal{N}$, *monotone* if $f(S) \leq f(T)$ for any $S \subseteq T \subseteq \mathcal{N}$ and *submodular* if $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ for any $S, T \subseteq \mathcal{N}$. Intuitively, a submodular function is a function that obeys the property of diminishing returns, i.e., the marginal

This work was supported in part by the National Natural Science Foundation of China (12471306), and the Fundamental Research Funds for the Central Universities.

D. Du et al. (Eds.): COCOA 2024, LNCS 15435, pp. 36–49, 2025.
https://doi.org/10.1007/978-981-96-4448-3_4

contribution of adding an element to a set diminishes as the set becomes larger and larger. Given a submodular function $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}^+$ and a family $\mathcal{F} \subseteq 2^{\mathcal{N}}$ of subsets of $\mathcal{N}$, the submodular maximization problem consists in finding a set $S \in \mathcal{F}$ that maximizes $f(S)$. A classic choice for $\mathcal{F}$ is the cardinality constraint where every subset $S$ of cardinality at most $k$ is feasible. For the submodular maximization problem under the cardinality constraint, the celebrated greedy algorithm due to Nemhauser and Wolsey [24] achieves an approximation ratio of $1 - 1/e$, which is optimal assuming $P \neq NP$. However, this classic algorithm is inefficient when applied to modern big data settings, given the unique challenges of working with massive datasets. Motivated by these challenges, in recent years there has been a surge of interest in considering the submodular maximization problem under a variety of computational models such as streaming models [2, 17]. The streaming algorithms can process the data in a streaming fashion, where only a small fraction of the data is kept in memory at any point.

**Submodular Maximization in Dynamic Model.** One fundamental limitation of streaming algorithms is that they are not well-suited to handle highly dynamic datasets, where elements are added and deleted continuously. Similarly, TikTok processes millions of video uploads and deletions each day, while also Snapchat processes millions of message uploads and deletions daily. For these reasons, many problems have been studied in the dynamic setting, even if it is notoriously difficult to obtain efficient algorithms. In the dynamic model, we are given a stream of insertions and deletions of elements of the underlying ground set $\mathcal{N}$ and we need to maintain a solution after every update. Thus, the update time of the dynamic algorithm should be as fast as possible. The goal is to maintain a good approximate set subject to the certain constraint after every insertion and deletion, which uses a fast query complexity.

For dynamic monotone submodular maximization with the cardinality constraint, a $(2 + \varepsilon)$-approximation algorithm with poly-logarithmic amortized update time was designed by Lattanzi et al. [19]. Subsequently, this result has been proved to be tight by Chen and Peng [11]. And they proved that developing a $c$-approximation dynamic algorithm for $c < 2$ is not possible unless we use a number of oracle queries polynomial in the size of ground set $\mathcal{N}$. Monemizadeh [23] achieved the same approximation guarantee with $\tilde{O}(k^2)$ amortized update time, where $k$ is the cardinality constraint. Then, Banihashem et al. [3] also improved the algorithm of [23] for monotone submodular maximization subject to the cardinality constraint. For the non-monotone submodular maximization problem, Banihashem et al. [4] studied the generalized version of the problem and obtained a dynamic algorithm that maintains a $(8 + \varepsilon)$-approximate solution per update. Recently, Dütting et al. [13] proposed the first fully dynamic algorithm for submodular maximization under the matroid constraint. And they obtained a randomized dynamic 4-approximation algorithm with expected amortized time $\tilde{O}(k^2)$ per update.

**Maximizing Submodular Functions Under the $p$-Matchoid Constraint.** The study of submodular maximization in the streaming setting has been mostly surveyed. Several works have considered maximization of both monotone and non-monotone submodular functions subject to the $p$-matchoid constraint. And $p$-matchoid generalizes many basic combinatorial constraints such as the cardinality constraint, the intersection of $p$ matroids, and matchings in graphs and hyper-graphs. When the objective function is monotone, the work of Chakrabarti and Kale [7] gave a $4p$-approximation streaming algorithm for maximizing such functions subject to the intersection of $p$ matroids constraints. Then the results were later extended by Chekuri et al. [8] to the $p$-matchoid constraint.

For non-monotone submodular objectives, the first streaming result was obtained by Buchbinder et al. [5]. They described a randomized streaming algorithm achieving a 11.197-approximation for the problem of maximizing a non-monotone submodular function subject to the cardinality constraint. Then, Chekuri et al. [8] described an algorithm of the same constraints achieving $(5p + 2 + 1/p)/(1 - \varepsilon)$-approximation for the problem of maximizing a non-monotone submodular function subject to the $p$-matchoid constraint. Simultaneously, they gave a deterministic streaming algorithm achieving $(9p+O(\sqrt{p}))/(1-\varepsilon)$-approximation for the problem. Mirzasoleiman et al. [22] came up with a different deterministic algorithm for the same problem achieving an approximation ratio of $4p + 4\sqrt{p} + 1$. Finally, Feldman et al. [14] described a new randomized streaming algorithm for maximizing a submodular function subject to the $p$-matchoid constraint and obtained an improved approximation ratio of $4p+2-o(1)$. The randomized streaming algorithm only used $O(k)$ memory and $O(km/p)$ value and independence oracle queries (in expectation) per element of the stream, which is even less than the number of oracle queries used by the state-of-the-art algorithm for monotone submodular objectives.

**Our Contribution.** In this paper, we design an efficient fully dynamic algorithm for monotone submodular maximization under the $p$-matchoid constraint. Our randomized algorithm processes a stream of arbitrarily interleaved insertions and deletions with an expected amortized time per update that is $O(qk^2 \log k \log^2 \Delta \log n)$, where $q$ is the number of matroids used to define the $p$-matchoid, $k$ is the size of the largest independent set of the same matroids, and $\Delta$ is a parameter that depends on the values of the set function. Crucially, it also continuously maintains a solution whose value is, after each update, at least $4p$ of the optimum on the available elements.

## 2 Preliminaries

Given a set $S$ and an element $e$, we denote by $S + e$ and $S - e$ the union $S \cup \{e\}$ and the expression $S \setminus \{e\}$, respectively. Additionally, the *marginal contribution* of $e$ to the set $S$ under the set function $f$ is written as $f_S(e) = f(S + e) - f(S)$. Using this notation, we can now state the above mentioned equivalent definition of submodularity, which is that a set function $f$ is *submodular* if and only if

$f_S(e) \geq f_T(e)$ for any two sets $S \subseteq T \subseteq \mathcal{N}$ and any element $e \in \mathcal{N} \setminus T$. Given two sets $S, T \subseteq \mathcal{N}$, we also refer to the *marginal gain* of $S$ with respect to $T$, $f_T(S)$, quantifies the change in value of adding $S$ to $T$ and is defined as $f_T(S) = f(S \cup T) - f(T)$.

**Matroids.** A set system is a pair $(\mathcal{N}, \mathcal{I})$, where $\mathcal{N}$ is the ground set of the set system and $\mathcal{I} \subseteq 2^{\mathcal{N}}$ is the set of independent sets of the set system. A *matroid* $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ is a set system which obeys the following three properties.

(1) $\emptyset \in \mathcal{I}$.
(2) *Downward-closure*: if $S \subseteq T \subseteq \mathcal{N}$ and $T \in \mathcal{I}$, then $S \in \mathcal{I}$.
(3) *Augmentation*: if $S, T \in \mathcal{I}$ with $|S| < |T|$, then there exists an element $e \in T \setminus S$ such that $S + e \in \mathcal{I}$.

The bases of $\mathcal{M}$ share a common cardinality, called the *rank* of the matroid $\mathcal{M}$. With the concept of matroids, we can generalize to define the $p$-matchoid.

**Matchoids.** Let $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_q$ be $q$ ground sets and $\mathcal{M}_1 = (\mathcal{N}_1, \mathcal{I}_1), \ldots, \mathcal{M}_q = (\mathcal{N}_q, \mathcal{I}_q)$ be $q$ matroids over overlapping ground sets. Let $\mathcal{N} = \mathcal{N}_1 \cup \cdots \cup \mathcal{N}_q$ and $\mathcal{I} = \{S \subseteq \mathcal{N} : S \cap \mathcal{N}_h \in \mathcal{I}_h \text{ for all } h\}$. The finite set system $\mathcal{M}^p = (\mathcal{N}, \mathcal{I})$ is a *p-matchoid* if for every element $e \in \mathcal{N}$, $e$ is a member of $\mathcal{N}_h$ for at most $p$ indices $h \in [q]$.

A simple example for a 2-matchoid is $\boldsymbol{b}$-matching. Recall that a set $E$ of edges of a graph is a $\boldsymbol{b}$-matching if and only if every vertex $v$ of the graph is hit by at most $b(v)$ edges of $E$, where $\boldsymbol{b}$ is a vector function assigning integer values to the vertices. The corresponding 2-matchoid $\mathcal{M}$ has the set of edges of the graph as its ground set and a matroid for every vertex of the graph. Let $\mathcal{M}_v = (\mathcal{N}_v, \mathcal{I}_v)$ be the matroid of a vertex $v$ of the graph, and its ground set $\mathcal{N}_v$ consists of the edges hitting $v$ and independent set is $\mathcal{I}_v = \{E_v \subseteq \mathcal{N}_v : |E_v| \leq b(v)\}$. Since every edge hits only two vertices, it appears in the ground sets of only two vertex matroids, thus $\mathcal{M}$ is a 2-matchoid. Moreover, one can verify that a set of edges is independent in $\mathcal{M}$ if and only if it is a valid $\boldsymbol{b}$-matching.

**Incremental Value.** Let $\mathcal{N}$ be a ground set, and let $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ be a submodular function over the ground set $\mathcal{N}$. For a set $S \subseteq \mathcal{N}$ and an element $e \in \mathcal{N}$, the *incremental value* of $e$ in $S$, denoted $f(e : S)$, is defined as

$$f(e : S) = f_{S'}(e) \text{ where } S' = \{s \in S : s < e\}.$$

Additionally, we denote by $e_1, e_2, \ldots, e_n$ the elements of $\mathcal{N}$ in the order in which they arrive. And $S'$ represents all the elements that were already in the solution $S$ before an element $e$ arrived. For an element $e \in \mathcal{N}$ and a set $S \subseteq \mathcal{N}$, then $f(S) = \sum_{e \in S} f(e : S)$. Intuitively, $f(e : S)$ is the marginal contribution of $e$ to the part of $S$ that arrived before $e$ itself.

**Notations for Dynamic Streams.** Consider a stream of exactly $n$ insertion and deletion operations chosen by an oblivious adversary. Denote by $\mathcal{N}_i$ the set of all elements that have been inserted and not deleted up to the $i$-th operation. Let $O_i$ be an optimum solution for $\mathcal{N}_i$, denote $OPT_i = f(O_i)$. An algorithm is a $\alpha$-approximation of the best (dynamic) solution if $OPT_i \leq \alpha f(S_i)$, for all $i = 1, 2, \cdots, n$, where $S_i \subseteq \mathcal{N}_i$ is a good feasible solution at the end of each operation $i$. In our dynamic algorithm, we are interested in updating our data structure efficiently. We say that an algorithm has amortized update time $t$ if its total running time to process a worst-case sequence of $n$ insertions and deletions is at most $nt$ in expectation.

**Maximizing a Monotone Submodular Function Under the $p$-Matchoid Constraint in Dynamic Model.** Let $\mathcal{N}$ be a set of elements, $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ be a non-negative monotone submodular function on $\mathcal{N}$, and $\mathcal{M}^p = (\mathcal{N}, \mathcal{I})$ be a $p$-matchoid for some integer $p$. Consider a stream of insertions and deletions. Let $\mathcal{N}_i$ be the set of all elements that have been inserted and not deleted up to the $i$-th operation. That is, $\mathcal{N}_i$ is the current ground set of elements, and $\mathcal{M}_i^p = (\mathcal{N}_i, \mathcal{I})$ is a $p$-matchoid for some integer $p$. Fix any operation $i$, the objective of our dynamic algorithm is to maintain a set $S_i \in \mathcal{I}$ which approximately maximizes $f$ at the end of the computation relative to operation $i$.

$$\max_{S_i \subseteq \mathcal{N}_i} \quad f(S_i)$$

$$\text{s.t.} \quad S_i \in \mathcal{I}$$

We want to approximate $OPT_i = \max_{S_i \in \mathcal{I}} f(S_i)$ at the end of each operation $i$. Throughout the paper, we assume that $f$ is normalized, i.e., $f(\emptyset) = 0$. We also assume that $f$ is given in terms of a value oracle that computes $f(S_i)$ for given $S_i \subseteq \mathcal{N}_i$, and independence oracles for the matroids defining the $p$-matchoid constraint. As usual in the field, we refer to running time as the total number of submodular function evaluations (value oracle) and independent set evaluations with respect to the $p$-matchoid (independence oracle). The number of non-oracle-call operations we perform is within a polylog factor of the number of oracle calls. We focus on the number of oracle calls performed during the computation and on the quality of returned solutions. More specifically, we perform a sequence of insertions and removals of elements, and after each operation $i$ we output a high-value set $S_i \in \mathcal{I}$.

Our main results for the dynamic setting are given by the following theorem. Recall that $k$ is the size of the largest independent set and $q$ is the number of matroids used to define the $p$-matchoid constraint. Throughout this paper, we assume value oracle to $f$ and independence oracles for each of the $q$ matroids defining the $p$-matchoid constraint.

**Theorem 1.** *Algorithm 4 yields a $4p$-approximation to the fully dynamic monotone submodular maximization problem with the $p$-matchoid constraint. And it exhibits an $O(qk^2 \log k \log^2 \Delta \log n)$ expected amortized running time.*

## 3 The Algorithm

In this section we provide an overview of the main techniques and ideas used in our dynamic algorithm. We start by noting that previous approaches either do not support deletions altogether, or support only a limited number of deletions and so they do not capture many real-world scenarios. In this work, we overcome this barrier by designing a novel fully dynamic data structure that has only poly-logarithmic amortized update time.

Our data structure contains $L+1$ levels, with $L = \log n$, and $n$ is the number of insertions or deletions of elements. Each one of these levels is characterized by three sets of elements: a partial solution $S_l$, a candidate elements set $A_l$ that meets certain criteria and is considered a good addition to the solution, and a buffer of still not processed elements $B_l$.

**Algorithm 1.** Initialization

```
Input: $n$.
Output: $\Theta(\log n)$ empty sets.
1: $L \longleftarrow \log n$
2: Initialize empty sets $S_l$, $A_l$, $B_l$   $\forall\ 0 \leq l \leq L$
```

The structure of each level of $A_l$ is essentially the same. The main difference is that different levels maintain different numbers of elements, i.e., level $l$ maintains $O(\frac{n}{2^l})$ many elements. Intuitively, and informally, levels with small $l$ are recomputed only after many updates. While levels with large $l$, such as $l = L$, are sensitive to update and recompute more frequently. In particular, if we insert an extremely valuable element, then the level $l = L$ will guarantee that this newly added valuable element will appear in the current solution $S$. We claim that the solution of the last level, i.e., $S_L$ (that plays the role of $S_i$), is consistently a constant factor approximation of $OPT_i$, at the end of each operation $i$.

**Initialization.** Consider a stream of exactly $n$ insertion and deletion operations. At the beginning of the stream, the routine Algorithm 1 is called. Specifically, $S_l$, $A_l$ and $B_l$ are set to empty sets, for all $l = 0, 1, \ldots, L$.

**Insert Elements.** When a new element $e$ is inserted, it gets immediately added to all the buffers $B_l$, for $0 \leq l \leq L$. Roughly speaking, our algorithm postpones processing insertions into level $l$ until there are $\frac{n}{2^l}$ many of them. This enables us to obtain efficient amortized update time of the structure on level $l$. The buffer set $B_l$ is used to store these insertions until they are processed. This addition induces the call of another routine, Level-Construct, on the level $l^\star$ with smallest index such that the buffer $B_l$ exceeds a certain cardinality.

**Algorithm 2.** Insertion($e$)

```
Input: the element $e$ to be inserted.
Output: a feasible solution $S$ after the element $e$ is inserted.
1: $B_l \longleftarrow B_l + e \quad \forall\, 0 \leq l \leq L$
2: $\mathcal{N} \longleftarrow \mathcal{N} + e$
3: if there exists an index $l$ such that $|B_l| \geq \frac{n}{2^l}$ then
4:     Let $l^\star$ be the smallest such index
5:     Level-Construct($l^\star$)
6: end if
7: $S \longleftarrow S_L$
```

**Delete Elements.** When an element $e$ is deleted from the stream, then the data structure is updated according to Algorithm 3. Element $e$ is removed from all the candidate elements sets $A_l$ and buffers $B_l$ (lines 1 and 2) and causes a call of Level-Construct on the smallest-index level such that $e \in S_l$.

**Algorithm 3.** Deletion($e$)

```
Input: the element $e$ to be deleted.
Output: a feasible solution $S$ after the element $e$ is deleted.
1: $A_l \longleftarrow A_l - e \quad \forall\, 0 \leq l \leq L$
2: $B_l \longleftarrow B_l - e \quad \forall\, 0 \leq l \leq L$
3: $\mathcal{N} \longleftarrow \mathcal{N} - e$
4: if $e \in S_l$ for some $l$ then
5:     Let $l^*$ be the smallest such index such that $e \in S_l$
6:     Level-Construct($l^*$)
7: end if
8: $S \longleftarrow S_L$
```

**Level-Construct.** We now describe the main routine of our data structure Level-Construct($l$). A call to this routine at level $l$ triggers some operations relevant to sets at level $l$, and it then recursively runs Level-Construct at level $l+1$. Therefore, Level-Construct($l$) is essentially responsible for reprocessing the whole data structure at all levels $l, l+1, \ldots, L$. When it is called on some level $l$, all the sets associated to that level $S_l, A_l$ and $B_l$ are reinitialized: the candidate elements set $A_l$ is initialized with the elements in $A_{l-1}$ and $B_{l-1}$, the buffer $B_l$ is erased, while $S_l$ is copied from the previous level. And we want to ensure that the subscripts of $A_l$ and $B_l$ are non-negative (lines 3-7 of Algorithm 4). Then, while the cardinality of $A_l$ becomes larger or equal to $\frac{n}{2^l}$, the following iterative procedure is repeated.

For any element $e' \in A_l$, we use a procedure called Swapping($S, e'$) to select a set $C_{e'}$. The set $C_{e'}$ consists of elements whose removal from the current solution maintained by the algorithm allows the addition of $e'$ to this solution. If

**Algorithm 4.** Level-Constrcut($l$)

```
Input: two non-negative parameters $\alpha$, $\beta$, and a call to this routine at level $l$.
Output: a feasible solution $S_L$.
 1: $B_l \longleftarrow \emptyset$
 2: $S_l \longleftarrow S_{l-1}$
 3: if $l > 0$ then
 4:     $A_l \longleftarrow A_{l-1} \cup B_{l-1}$
 5: else
 6:     $A_l \longleftarrow \mathcal{N}$
 7: end if
 8: while $|A_l| \geq \frac{n}{2^l}$ do
 9:     for any element $e' \in A_l$ do
10:         $C_{e'} \longleftarrow$ Swapping$(S_l, e')$
11:     end for
12:     $A_l \longleftarrow \Big\{e' \in A_l \mid f_{S_l}(e') \geq \alpha + (1+\beta)\sum_{c \in C_{e'}} f(c : S_l)\Big\}$
13:     if $|A_l| \geq \frac{n}{2^l}$ then
14:         Pop $e'$ from $A_l$ uniformly at random
15:         $S_l \longleftarrow S_l \setminus C_{e'} + e'$
16:     end if
17: end while
18: if $l < L$ then
19:     Level-Construct$(l+1)$
20: end if
```

**Algorithm 5.** Swapping($S, e'$)

```
Input: the current solution $S$, an element $e' \in A_l$.
Output: $C$.
 1: $C \longleftarrow \emptyset$
 2: for $h = 1, \ldots, q$ do
 3:     if $e' \in \mathcal{N}_h$ and $(S + e') \cap \mathcal{N}_h \notin \mathcal{I}_h$ then
 4:         $S_h = S \cap \mathcal{N}_h$
 5:         $X_h \longleftarrow \{s \in S_h : (S_h - s + e') \in \mathcal{I}_h\}$
 6:         $c_h \longleftarrow \arg\min_{x \in X_h} f(x : S)$
 7:         $C \longleftarrow C + c_h$
 8:     end if
 9: end for
```

the marginal contribution of adding $e'$ to the solution is large enough compared to the value of the elements of $C_{e'}$, then $e'$ is added to the solution and the elements of $C_{e'}$ are removed. All in all, Algorithm 4 maintains an independent set $S_l \in \mathcal{I}$, for $0 \leq l \leq L$. All the elements in $A_l$, either discarded or added to $S_l$ in exchange for a well-chosen subset of $S_l$, for $0 \leq l \leq L$. The threshold for exchanging is tuned by two non-negative parameters $\alpha$ and $\beta$ (line 12 of Algorithm 4). Then, if the cardinality of $A_l$ is still large enough (i.e., $|A_l| \geq \frac{n}{2^l}$) an element $e'$ is added to $S_l$ and the elements of $C_{e'}$ are removed.

## 4 Analysis of the Algorithm

In this section, we state the analysis of our dynamic algorithm. We consider the approximation guarantee and the amortized running time of our algorithm, respectively. Fix any operation $i$, we want to show that $S_i$ is a good approximation of the best independent set $O_i$ ($f(O_i) = OPT_i$) in $\mathcal{N}_i$ at the end of operation $i$.

### Some Notations for the Analysis

- $S_L$ : the final output of Algorithm 4.
- For each element $e \in \mathcal{N}$, $S^{-}_{l,e}$ denotes the set $S_l$ just before $e$ is processed on level $l$, and $S^{+}_{l,e}$ the set $S_l$ just after $e$ is processed on level $l$. Note that if $e$ is rejected on level $l$, then $S^{-}_{l,e} = S^{+}_{l,e}$.
- $D$ denotes the set of all elements ever added to $S_l$, for all $0 \leq l \leq L$.
- For $e \in \mathcal{N}$, $C_e(=\text{Swapping}(S^{-}_{l,e}, e) \subseteq S^{-}_{l,e})$ denotes the set of elements that Level-Construct($l$) considers exchanging for $e$. Observe that $\{C_d : d \in D\}$ forms a partition of $D \setminus S_L$.
- For $e \in \mathcal{N}$, $\delta_e = f(S^{+}_{l,e}) - f(S^{-}_{l,e})$ denotes the gain from processing $e$ by Level-Construct($l$). Note that $\delta_e = 0$ for all $e \in \mathcal{N} \setminus D$, and $\sum_{e \in \mathcal{N}} \delta_e = f(S_L)$.

The first lemma derives a lower bound for this gain.

**Lemma 1.** *Let $e' \in D$ be added to $S_l$ when processed by Level-Construct($l$). Then*

$$\delta_{e'} \geq \alpha + \beta \sum_{c \in C_{e'}} f(c : S^{-}_{l,e'}).$$

To relate the final output $S_L$ to $OPT$, we consider $D$, the set of all elements ever added to $S_l$, for all $0 \leq l \leq L$. For $d \in D \setminus S_L$, let $e'(d)$ be the element that $d$ was exchanged for, that is, $e'(d)$ appeared later than $d$ in the stream and $d \in C_{e'(d)}$. For deleted elements $d \in D \setminus S_L$, the *exit value* $\mu(d)$ of $d$ is the incremental value of $d$ evaluated when $d$ is removed from $S_l$, defined formally as

$$\mu(d) = f(d : S^{-}_{l,e'(d)}).$$

The analysis proceeds in two steps. First, we bound $f(D)$ in the following lemma.

**Lemma 2.** *Let $D$ be the set of all elements ever added to $S_l$, for all $0 \leq l \leq L$. Then*

$$f(D) \leq (1 + \frac{1}{\beta}) \cdot f(S_L) - \frac{\alpha}{\beta} |D| .$$

*Proof.* For each element $d \in D \setminus S_L$, that is, $d$ is an element that was once in the solution $S_l$ on some level $l$ but has been replaced later. And $e'(d)$ denotes the element added in exchange of $d$. We have

$$\begin{aligned} f(D) - f(S_L) \le f_{S_L}(D) &= \sum_{d \in D \setminus S_L} f_{S_L}(d : D) \\ &\le \sum_{d \in D \setminus S_L} \mu(d) \\ &= \sum_{e' \in D} \sum_{d \in C_{e'}} \mu(d) \\ &\le \sum_{e' \in D} \frac{1}{\beta} \cdot (\delta_{e'} - \alpha) \\ &= \frac{1}{\beta} \cdot f(S_L) - \frac{\alpha}{\beta} \left| D \right|, \end{aligned}$$

where the first inequality holds since the monotonicity of $f$, the second inequality holds since the submodularity of $f$ and the definition of $\mu(d)$ and the last inequality follows by Lemma 1.

Second, we upper bound $f(M \cup D)$. Let $M \in \mathcal{I}$ be any feasible solution. To relate the final output $S_L$ to $OPT$, we upper bound $f(M \cup D)$. Here we use the fact that $\mathcal{I}$ is a $p$-matchoid to frame an exchange argument between $M$ and $D$.

**Lemma 3.** *Let $M \in \mathcal{I}$ be an any independent set. Then*

$$\mathbf{E}[f(M \cup D)] \le k\alpha + \frac{(1+\beta)^2}{\beta} \cdot p \cdot \mathbf{E}[f(S_L)].$$

**Theorem 2.** *Let $\mathcal{M}^p = (\mathcal{N}, \mathcal{I})$ be a $p$-matchoid of rank $k$, and let $f : 2^{\mathcal{N}} \longrightarrow \mathbb{R}^+$ be a non-negative monotone submodular function. For any operation $i$, it holds that the solution $S_i$ output by Algorithm 4 at the end of the computation relative to iteration $i$ is a $4p$-approximation of $OPT_i$, for $\alpha = 0, \beta = 1$.*

*Proof.* Consider the last time $i$ when Level-Construct($l$) is called for some value $i$. Let us start by analyzing $f(S_i)$. If $f$ is monotone, then $f(M) \le f(M \cup D)$ for any set $M$. Let $\alpha = 0, \beta = 1$, $S_i$ be the solution of our algorithm and $O_i$ be the optimal solution after $i$ updates. By Lemma 3, we obtain the followings.

$$\mathbf{E}[f(S_i)] \ge \frac{1}{4p} \cdot \mathbf{E}[f(M \cup D)].$$

If we take $M$ to be the set achieving $O_i$, then

$$\mathbf{E}[f(S_i)] \ge \frac{1}{4p} \cdot \mathbf{E}[f(O_i \cup D)] \ge \frac{1}{4p} \cdot f(O_i) = \frac{1}{4p} \cdot OPT_i.$$

Now we prove the amortized running time of Algorithm 4. Our data structure whose amortized running time depends poly-logarithmically on a parameter $\Delta$ of the function $f$ defined as

$$\Delta = \frac{\max_{x \in \mathcal{N}} f(x)}{\min_{T \subseteq \mathcal{N},\ x \notin T_0} f_T(x)},$$

where $T_0$ denotes the set of all the elements with 0 marginal contribution with respect to $T$. Recall that, throughout this paper, we refer to running time as the total number of the submodular function oracle evaluations plus the number of the $p$-matchoid independent set oracle evaluations.

First, we show that it is possible to compute the candidate swap $C_{e'}$ in line 10 of Algorithm 4 in $O(q \log k)$ calls of the independence oracle of the $p$-matchoid.

**Lemma 4.** *For any element $e' \in A_l$, it is possible to find the candidate swap $C_{e'}$ by Swapping($S, e'$) in $O(q \log k)$ calls of the independence oracle of the $p$-matchoid.*

The computation of our dynamic algorithm is mainly carried out by the Level-Construct function, which exhibits a recursive structure. While Insertion($e$) or Deletion($e$) triggers Level-Construct($l$), this induces a chain of recursive calls of Level-Construct to higher levels. We construct a deterministic upper bound on the computation induced by any new chain of Algorithm 4 calls in Lemma 5.

**Lemma 5.** *For any level $l$ with $0 \leq l \leq L$, the running time of Level-Construct($l$) is $O(\frac{nkq \log \Delta \log k}{2^l})$.*

Then, we measure the number of times that Insertion($e$) induce new chain of Algorithm 4 calls by Lemma 6.

**Lemma 6.** *For any level $l$ with $0 \leq l \leq L$, the number of invocations that the Level-Construct($l$) is called directly from Insertion($e$) is at most $2^l$.*

Next, we measure the number of times that Deletion($e$) induce new chain of Algorithm 4 calls by Lemma 7.

**Lemma 7.** *For any level $l$ with $0 \leq l \leq L$, the expected number of invocations that the Level-Construct($l$) is called directly from Deletion($e$) is at most $2^{l+3} k \log \Delta$.*

Combining the upper bound of Algorithm 4 with the number of calls completes the proof of Theorem 3.

**Theorem 3.** *Algorithm 4 exhibits an $O(qk^2 \log k \log^2 \Delta \log n)$ expected amortized running time.*

*Proof.* The running time when an element is inserted or deleted is $O(L) = O(\log n)$ beside the calls made to Level-Construct. By comparing Lemma 6 and

Lemma 7, it results that the number of calls induced by Deletion$(e)$ dominates those induced by Insertion$(e)$. We bound the total expected running time by

$$\begin{aligned}
&2 \sum_{0 \le l \le L} k 2^{l+3} \log \varDelta \cdot (c \cdot \frac{nkq \log k \log \varDelta}{2^l}) \\
&= 16c \sum_{0 \le l \le L} nqk^2 \log k \log^2 \varDelta \\
&= 16c \cdot (nqk^2 \log k \log^2 \varDelta \log n).
\end{aligned}$$

## 5 Conclusion

In this paper, we study monotone submodular maximization with the $p$-matchoid constraint in the dynamic setting. And we construct a dynamic algorithm whose guarantees are robust to any adversary that generates the stream of insertions and deletions. Our main result is a randomized algorithm that obtains a $4p$-approximate solution with only poly-logarithmic amortized update time. Specifically, our dynamic algorithm exhibits an $O(qk^2 \log k \log^2 \varDelta \log n)$ expected amortized running time, where $k$ is an upper bound on the cardinality of the feasible solution. For future work, it would be interesting to extend the current result to more general constraints. Another compelling direction for future work is to reduce the amortized running to depend only poly-logarithmically in $k$.

## References

1. Babaioff, M., Immorlica, N., Kleinberg, R.: Matroids, secretary problems, and online mechanisms. In: Proceedings of the 18th Symposium on Discrete Algorithms, pp. 434–443 (2007)
2. Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.: Streaming submodular maximization: massive data summarization on the fly. In: Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining, pp. 671–680 (2014)
3. Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., Monemizadeh, M.: Dynamic algorithms for matroid submodular maximization. In: Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, pp. 3485–3533 (2024)
4. Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., Monemizadeh, M.: Dynamic non-monotone submodular maximization. In: Proceedings of the 36th Neural Information Processing Systems, pp. 17369–17382 (2023)
5. Buchbinder, N., Feldman, M., Schwartz, R.: Online submodular maximization with preemption. ACM Trans. Algorithms **15**(3), 1–31 (2019)
6. Calinescu, G., Chekuri, C., Pál, M., Vondrák, J.: Maximizing a submodular set function subject to a matroid constraint. In: Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization, pp. 182–196 (2007)

7. Chakrabarti, A., Kale, S.: Submodular maximization meets streaming: matchings, matroids, and more. Math. Program. **154**, 225–247 (2015). https://doi.org/10.1007/s10107-015-0900-7
8. Chandra C., Shalmoli G., Kent Q.: Streaming algorithms for submodular function maximization. In: Proceedings of the 42nd International Colloquium on Automata, Languages and Programming, pp. 318–330 (2015)
9. Chen, W., Wang, C. Wang, Y.: Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Proceedings of the 16th International Conference on Knowledge Discovery and Data Mining, pp. 1029–1038 (2010)
10. Chen, W., Wang,Y., Yang, S.: Efficient influence maximization in social networks. In: Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining, pp. 199–208 (2009)
11. Chen, X., Peng, B.: On the complexity of dynamic submodular maximization. In: Proceedings of the 54th Symposium on Theory of Computing, pp. 1685–1698 (2022)
12. Dasgupta, A., Kumar, R., Ravi, S.: Summarization through submodularity and dispersion. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pp. 1014–1022 (2013)
13. Dütting, P., Fusco, F., Lattanzi, S.: Fully dynamic submodular maximization over matroids. In: Proceedings of the 40th International Conference on Machine Learning, pp. 8821–8835 (2023)
14. Feldman, M., Karbasi, A., Kazemi, E.: Do less, get more: streaming submodular maximization with subsampling. In: Proceedings of the 31st Neural Information Processing Systems, pp. 730–740 (2018)
15. Fujishige, S.: Submodular Functions and Optimization, vol. 58. Elsevier (2005)
16. Goyal, A., Bonchi, F., Lakshmanan, L.: A data-based approach to social influence maximization. Proce. VLDB Endow. **5**(1), 73–84 (2011)
17. Kazemi, E., Mitrovic, M., Zadimoghaddam, M., Lattanzi, S., Karbasi, A.: Submodular streaming in all its glory: tight approximation, minimum memory and low adaptive complexity. In: Proceedings of the 36th International Conference on Machine Learning, pp. 3311–3320 (2019)
18. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining, pp. 137–146 (2003)
19. Lattanzi, S., Mitrovic, S., Norouzi-Fard, A., Tarnawski, J., Zadimoghaddam, M.: Fully dynamic algorithm for constrained submodular optimization. In: Proceedings of the 33rd Neural Information Processing Systems, pp. 12923–12933 (2020)
20. Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., Glance, N.: Cost-effective outbreak detection in networks. In: Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining, pp. 420–429 (2007)
21. Lin, H., Bilmes, J.: A class of submodular functions for document summarization. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pp. 510–520 (2011)
22. Mirzasoleiman, B., Jegelka, S., Krause, A.: Streaming non-monotone submodular maximization: personalized video summarization on the fly. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
23. Monemizadeh M.: Dynamic submodular maximization. In: Proceedings of the 33rd Neural Information Processing Systems, pp. 9806–9817 (2020)
24. Nemhauser, G.L., Wolsey, L.A.: Best algorithms for approximating the maximum of a submodular set function. Math. Oper. Res. **3**(3), 177–188 (1978)

25. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency, vol. 24, Springer (2003)
26. Seeman , L., Singer, Y.: Adaptive seeding in social networks. In: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science, pp. 459–468 (2013)
27. Sipos, R., Swaminathan, A., Shivaswamy, P., Joachims, T.: Temporal corpus summarization using submodular word coverage. In: Proceedings of the 21st International Conference on Information and Knowledge Management, pp. 754–763 (2012)

# Generative Flow Networks with Symmetry Enhancement to Solve Vehicle Routing Problems

Zizhen Zhang[1], Guoyao Rao[2], Deying Li[1](✉), Yongcai Wang[1], Wenping Chen[1], and Yuqing Zhu[3]

[1] Renmin University of China, Beijing, Beijing 100872, China
{zizhenzhang,deyingli,ycw,chenwenping}@ruc.edu.cn
[2] Department of New Networks, Peng Cheng Laboratory, Shenzhen, China
raogy@pcl.ac.cn
[3] California State University, Los Angeles, USA
yuqing.zhu@calstatela.edu

**Abstract.** Vehicle Routing Problems (VRPs) have made significant strides in accuracy and computational efficiency through the adoption of Deep Learning (DL) techniques. However, previous studies have not fully addressed the symmetries inherent in VRPs, such as rotation, translation, permutation, and scaling. This paper introduces a novel training approach, GSE-VRPs, which employs a regularizer-based method to exploit universal symmetries present in various VRPs and their solutions. By leveraging symmetries like rotational, reflectional, and uniform scalability invariance, this approach substantially enhances the generalization capability of neural heuristic solvers. It enables learned solvers to effectively utilize common symmetries within the same class of VRPs. Our experiments demonstrate that GSE-VRPs significantly enhance the performance of deep heuristic methods in two VRP tasks—the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP)—all without relying on problem-specific domain expertise.

**Keywords:** Vehicle Routing Problems · Symmetry · Neural Heuristic

## 1 Introduction

Combinatorial optimization problems (COPs), characterized by NP-hardness, involve a discrete search space and computational challenges in finding the optimal solution. One key example of a COP in logistics is the vehicle routing problem (VRP), which focuses on minimizing delivery costs by efficiently routing vehicles from a depot to a set of geographically dispersed customers. The VRP has been the subject of extensive research for decades and has found numerous

Dr. Li is supported in part by the National Natural Science Foundation of China Grant No. 12071478.

real-world applications, including freight delivery [5], last-mile logistics [9], and ride-hailing services [15].

In recent years, research on solving the VRP using deep reinforcement learning (DRL) has been gaining momentum. Unlike traditional approaches, this emerging field focuses on automatically learning heuristic policies through neural networks, which can identify underlying patterns in problem instances. This allows the discovery of more effective policies than those created manually [4]. Significant efforts have been made to develop various deep learning models aimed at narrowing the performance gap between emerging methods and highly optimized conventional heuristic solvers like Concorde [1] and LKH [6]. These deep models target solving VRP variants, such as the traveling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP) [10,11,13,17].

Despite recent efforts to apply DRL to VRPs, these approaches have largely overlooked an important characteristic: symmetry inherent in VRPs. For instance, problems like the TSP, often depicted in 2D Euclidean space, require solutions that remain unchanged under geometric transformations of city coordinates, such as rotation, reflection, and translation. Moreover, DRL typically neglects solution symmetry, where an optimal Hamiltonian cycle in TSP can correspond to multiple equivalent sequences or trajectories in DRL's sequential decision-making framework. Utilizing symmetry in training neural networks is essential, as it serves as an inductive bias that helps reduce the training space, thereby enhancing the efficiency of the deep reinforcement learning (DRL) process.

Recent studies have made efforts to tackle the challenges encountered by earlier neural solvers for VRPs, but only from some specific angles. For instance, some researchers have concentrated on adjusting their REINFORCE baseline by incorporating terms that account for symmetry [8,11], while others have introduced RL-based heuristic enhancements that capitalize on the cyclic properties of the TSP [13]. Despite these efforts to address symmetry issues, they are often tailored to specific problems and typically result in only incremental improvements.

To overcome these challenges, we introduce a generative strategy based on a new flow-matching learning paradigm [3], a decision-making framework aimed at learning stochastic policies for sampling composite objects with probabilities that align with a given terminal reward. The foundational elements of our framework include a flow-match loss that reflects solution symmetry and a mean square error loss that addresses problem symmetry. Specifically, we approach VRPs as a sequential decision-making process, utilizing an autoregressive model to sequentially select locations to visit. The autoregressive location selection process is represented as a generation tree, with each potential route depicted as a trajectory from the root to a leaf. The generative model manages the probability flow throughout the tree to ensure that the distribution of probabilities across routes aligns with their respective utilities. Since the symmetric transformation problem and the original problem have the identical generation tree, we introduce a bias penalty term loss, which penalizes discrepancies between the flow values of original and symmetric transformed coordinate sets. We have tested GSE-VRPs on two classical VRPs, including the TSP and the CVRP. Our results demonstrate that GSE-VRPs delivers significant improvements over neural solvers trained using standard RL paradigms. We also performed ablation studies to validate the effectiveness of our approach. In summary, our main contributions are as follows:

- We tackle the issue of problem symmetry in constructive methods for solving vehicle routing problems, which often result in multiple optimal solutions. These symmetries can be leveraged during neural network training through symmetry alignment, enhancing the training process.
- We present an efficient learning algorithm, GFlowNet, designed to facilitate rapid credit assignment for GFlowNet agents navigating symmetrical trajectories commonly found in vehicle routing problems.
- We validate the efficiency of our model in the experiments which demonstrate that our model achieves superior generalization and stability.

## 2 Related Work

Bello et al. [2] introduced one of the earliest deep reinforcement learning (DRL) methods using a pointer network [16], trained through an actor-critic approach. The Attention Model (AM) [10] extends this work by substituting the Pointer-Net with a Transformer architecture, establishing itself as the standard for neural solvers. Notably, AM showcases its versatility by effectively addressing various classical routing problems and their practical extensions [5,7]. The multi-decoder Attention Model (MDAM) [18] builds upon the Attention Model by integrating an ensemble of decoders. However, this extension is not ideally suited for addressing stochastic routing problems. POMO [11] and Sym-NCO [8] enhance the Attention Model by exploiting symmetry in the TSP and the CVRP. These methods optimise in the direction of the best of the multiple trajectories generated by the neural solver, which can lead to a lack of exploration of the solution space, arriving at a local optimum rather than a global one.

## 3 Preliminary

### 3.1 Problem Formulation

In this work, we focus on 2D Euclidean Vehicle Routing Problems (VRPs). In this study, we concentrate on 2D Euclidean Vehicle Routing Problems (VRPs). The constrained VRP can be formulated as follows:

*Problem 1 (Vehicle Routing Problems (VRPs)).* Given a set of city coordinates $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subseteq \mathbb{R}^2$, the goal is to determine an optimal route $\tau =$

$\{\tau(1), \ldots, \tau(N)\}$ that minimizes the total distance $L$ traveled along the route. The objective function for the VRPs can be expressed as follows:

$$\begin{aligned} \min_{\tau} \quad & L(\tau, \mathbf{X}) = \sum_{i=1}^{N} \|\mathbf{x}_{\tau(i)} - \mathbf{x}_{\tau(i+1)}\|_2 \\ \text{s.t.} \quad & f(\tau, \mathbf{X}) = 0, \\ & g(\tau, \mathbf{X}) \leq 0. \end{aligned} \tag{1}$$

where $f(\tau, \mathbf{X})$ and $g(\tau, \mathbf{X})$ denote the constraint functions. For different VRP variants, the objective may be affected by various problem-specific constraints, allowing for multiple sub-routes within a valid route $\tau$. In accordance with recent literature [10,11], we focus on two representative VRP variants: the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP). In the TSP, a valid route is a Hamiltonian cycle in which each node is visited exactly once before returning to the starting point. In the CVRP, the depot $\mathbf{x}_0$ is part of the set $\mathbf{X}$, and each customer node $\mathbf{x}_i$ $(i \geq 1)$ has an associated demand $\delta_i$. A feasible solution consists of multiple sub-routes, with each vehicle departing from the depot, serving a subset of customers, and returning to the depot. Each customer node $\mathbf{x}_i$ $(i \geq 1)$ must be visited exactly once, and the total demand for each sub-route must not exceed the vehicle's capacity $\Delta$.

## 3.2 GFlowNets

In this section, we present the fundamental concepts of GFlowNets, a novel generative model designed for compositional objects denoted as $x \in \mathcal{X}$. We adhere to the notation established in [3]. GFlowNets utilize a trajectory-based generative process, wherein discrete actions iteratively alter a state that represents a partially constructed object. This process is illustrated by a directed acyclic graph (DAG), $G = (\mathcal{S}, \mathcal{A})$, where $\mathcal{S}$ represents the finite set of all possible states, and $\mathcal{A}$ denotes a subset of $\mathcal{S} \times \mathcal{S}$, corresponding to the directed edges that connect the states. The head of an edge corresponds to a state $s$, while the parents of state $s$ consist of the set of states connected by edges for which $s$ acts as the tail.

We define a complete trajectory as $\tau = (s_0 \to \cdots \to s_n) \in \mathcal{T}$, which progresses from the initial state $s_0$ to the terminal state $s_n = x \in \mathcal{X}$. The trajectory flow $F(\tau) : \mathcal{T} \to \mathbb{R}_{\geq 0}$ represents the unnormalized density function for a given trajectory $\tau \in \mathcal{T}$. The state flow is defined as the total unnormalized probability that passes through a state $s$: $F(s) = \sum_{\tau \in \mathcal{T}: s \in \tau} F(\tau)$. Similarly, the edge flow signifies the total unnormalized probability passing through an edge $s \to s'$: $F(s \to s') = \sum_{\tau \in \mathcal{T}: (s \to s') \in \tau} F(\tau)$. The trajectory reward $R(\tau)$ is defined as the reward associated with the terminal state of the trajectory, expressed as $R(\tau = (s_0 \to \cdots \to s_n = x)) = R(x)$. The forward policy, $P_F(s'|s)$, models the transition probability from state $s$ to its child state $s'$. Similarly, the backward policy, $P_B(s|s')$, models the transition probability for the reverse transition from $s'$ to its parent state $s$. The forward and backward policies,

$P_F$ and $P_B$, are related to the Markovian flow $F$ through the following equations: $P_F(s'|s) = \frac{F(s \to s')}{F(s)}$ and $P_B(s|s') = \frac{F(s' \to s)}{F(s')}$. The marginal likelihood of sampling $x \in \mathcal{X}$ is defined as $P_F^\top(x) = \sum_{\tau \in \mathcal{T}: \tau \to x} P_F(\tau)$, where $\tau \to x$ indicates a complete trajectory $\tau$ that concludes at $x$. The primary objective of GFlowNets is to align the marginal likelihood with the reward function, such that $P_F^\top \propto R(x) = \exp(-\mathcal{E}(x)/T)$, where $\mathcal{E} : \mathcal{X} \to \mathbb{R}$ represents an energy function, and $T > 0$ is a temperature parameter.

**Trajectory Balance (TB)** [14]**.** The trajectory balance loss $\mathcal{L}_{TB}$ operates by training three components: a learnable scalar for the initial state flow $Z_\theta \approx F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau)$, a forward policy $P_F(s_{t+1}|s_t; \theta)$, and a backward policy $P_B(s_t|s_{t+1}; \theta)$. These models are trained to minimize the following objective:

$$\mathcal{L}_{TB}(\tau; \theta) = \left( \log \frac{Z_\theta \prod_{i=0}^{n-1} P_F(s_{i+1}|s_i; \theta)}{R(x) \prod_{i=0}^{n-1} P_B(s_i|s_{i+1}; \theta)} \right)^2. \tag{2}$$

## 4 Method

### 4.1 Learning Constructive Heuristics for VRPs

The goal of VRPs is generally to identify a feasible solution $\tau = (a_1, a_2, a_3, \dots)$, where each $a_t$ corresponds to the information of the $t$-th selection, such as a location index for VRPs. While constructing the tour $\tau$ step by step, our model operates within a Markov Decision Process (MDP), described as follows:

- **State.** The state $s_t = (a_{1:t}, \mathbf{X})$ represents the partially constructed solution at step $t$, where $a_{1:t}$ indicates the locations that have been selected so far. The initial state $s_0$ corresponds to an empty solution, while the terminal state $s_N$ signifies the completed solution.
- **Action.** The action $a_t$ involves selecting a location from the set of unvisited nodes (i.e., $a_t \in \mathcal{A}_t = \{1, \dots, N\} \backslash \{a_{1:t-1}\}$).
- **Reward.** We define the log reward as the total length of a completed solution, given by $\mathcal{E}(a_{1:n}) = L(a_{1:n}, \mathbf{X})$.

For instance, let's consider solving the Traveling Salesman Problem (TSP) with four cities (i.e., $N = 4$). The problem instance $\mathbf{X}$ includes the coordinates of these four cities, represented as $\mathbf{X} = \{\mathbf{x}\}_{i=1}^4$, which the salesman needs to visit. The solution $\tau = a_{1:N}$ is a sequence of city indices. If $a_{1:N} = (1, 3, 2, 4)$, the salesman visits the cities in the order $\mathbf{x}_1 \to \mathbf{x}_3 \to \mathbf{x}_2 \to \mathbf{x}_4 \to \mathbf{x}_1$, returning to the first city to complete the route. The log reward of the TSP is given by $(\sum_{t=1}^3 ||\mathbf{x}_{a_{t+1}} - \mathbf{x}_{a_t}|| + ||\mathbf{x}_{a_4} - \mathbf{x}_{a_1}||)$.

Symmetries are intrinsic to many VRPs, and we hypothesize that leveraging these symmetries in the neural solver can improve both its generalization ability and sample efficiency. We define the two identified symmetries as follows (Fig. 1):

**Definition 1 (Problem Symmetry).** *Problem $\mathbf{X}^i$ and $\mathbf{X}^j$ are problem symmetric ($\mathbf{X}^i \xleftrightarrow{sym} \mathbf{X}^j$) if for any feasible solution $\tau$, $R(\tau; \mathbf{X}^i) = \alpha R(\tau; \mathbf{X}^j)$, where $\alpha \in \mathbb{R}_+$.*

**Definition 2 (Solution Symmetry).** *Two solution $\tau^i$ and $\tau^j$ are solution symmetric ($\tau^i \overset{sym}{\longleftrightarrow} \tau^j$) on problem $\mathbf{X}$ if $R(\tau^i; \mathbf{X}) = R(\tau^j; \mathbf{X})$.*

Several notable problem symmetries are found in VRPs, including rotation, reflection, and uniform scaling. In the two-dimensional coordinate plane, rotation and reflection can be represented by orthogonal matrices. In contrast, uniform scaling can be represented by diagonal matrices, with the diagonal elements being the scaling factors $\alpha$.

**Proposition 1 (Orthogonal Symmetry).** *For any orthogonal matrix $O$, the problem $\mathbf{X}$ and $O(\mathbf{X}) = \{O\mathbf{x}_i\}_{i=1}^N$ are problem symmetric, i.e., $\mathbf{X} \overset{sym}{\longleftrightarrow} O(\mathbf{X})$.*

**Proposition 2 (Uniform Scaling Symmetry).** *For the uniform scaling matrix $S^\alpha = \alpha \cdot I$, where $\alpha \in \mathbb{R}$ is the scaling factor, the problem $\mathbf{X}$ and $S(\mathbf{X}) = \{S\mathbf{x}_i\}_{i=1}^N$ are problem symmetric, i.e., $\mathbf{X} \overset{sym}{\longleftrightarrow} S(\mathbf{X})$.*

Set $O_{ro}$ to denote the rotation transformation matrix and $O_{re}$ to denote the reflection transformation matrix. We can combine these transformation matrices to get a composite transformation matrix. Assuming that we rotate, then reflect, and finally uniform scale in that order, the composite transform matrix ($M$) is:

$$M^\alpha = S^\alpha \cdot O_{re} \cdot O_{ro} \tag{3}$$

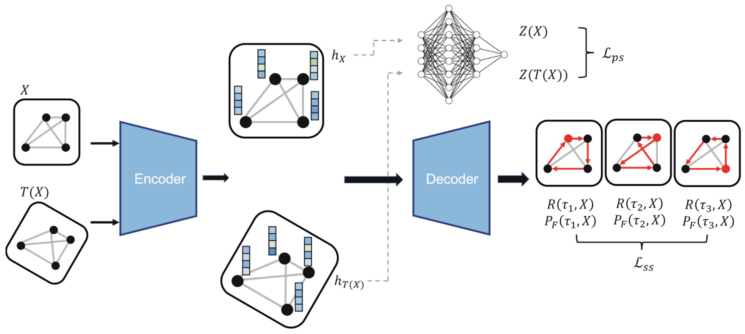


**Fig. 1.** The framework of GSE-VRPs.

## 4.2 GFlowNets with Symmetry Enhancement for VRPs

In this section, we present our proposed framework, the **G**enerate Flow Network with **S**ymmetry **E**nhancement for **V**ehicle **R**outing **P**roblems (GSE-VRPs). GSE-VRPs learns the neural solver parameters $\theta$ by minimizing the total loss function:

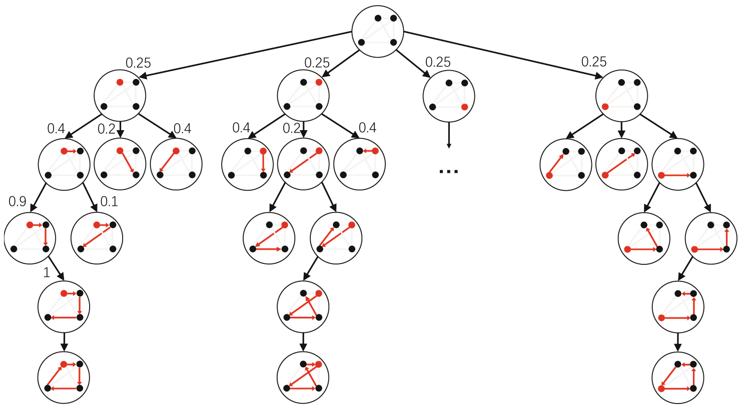


**Fig. 2.** Example of generation tree with $N = 4$ for TSP.

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{ss}} + \lambda_1 \mathcal{L}_{\text{ps}} \tag{4}$$

The total loss, $\mathcal{L}_{\text{total}}$, comprises $\mathcal{L}_{\text{ss}}$, the loss term from Eq. (4) that incorporates solution symmetry regularization; and $\mathcal{L}_{\text{ps}}$, the loss term from Eq. (4) that applies problem symmetry regularization. The weight coefficients $\lambda_1$ range from 0 to 1. In the following sections, we explain each loss term in detail.

## 4.3 Solution Symmetry $\mathcal{L}_{\text{ss}}$

Assuming the set of cities to be visited contains $N$ locations, the process of sequentially adding these locations to the route creates a generation graph that resembles a tree structure with a depth of $N$. In this tree, nodes represent intermediate or final routes, while each edge corresponds to a selected location. Figure 2 illustrates an example of a generation tree for the Traveling Salesman Problem (TSP) with $N = 4$. In this tree graph, each node $s_t$ has a single parent node $s_{t-1}$, except for the root node, which has no parent. The number of child nodes for a given node $s_t$ is proportional to $|N| - t$, except for leaf nodes, which do not have any children. All leaf nodes are located at depth $N$, and the total number of leaves (i.e., the route search space) is equivalent to the number of possible arrangements of the $N$ locations, which is $N!$. By sampling from the auto-regressive location selection model $P_F(s_t|s_{t-1}, \mathbf{X})$, the generator creates a trajectory that leads to the output route $\tau = a_1, \ldots, a_N$. Each output route (located at the leaf node) uniquely corresponds to its generation trajectory. Thus, the generation probability of an output route is determined by the sampling probability of its unique trajectory, conditioned on $\mathbf{X}$:

$$P_F(\tau|\mathbf{X}) = \prod_{t=1}^{N} P_F(s_t|s_{t-1}, \mathbf{X}) \tag{5}$$

where the selection of node $s_t$ determines the next step of the output route, i.e., $P_F(s_t|s_{t-1}, \mathbf{X})$. Using the example in Fig. 2, the route $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ has a trajectory probability $P_F(\tau|\mathbf{X}) = 0.25 \times 0.4 \times 0.9 \times 1 = 0.09$.

In contrast to the standard reward maximization objective typically seen in most neural solvers, our goal is to learn a diverse policy that not only identifies the highest reward but also fosters exploration by encouraging other high-reward routes. Consequently, in line with Eq. (2), our objective is to learn a trajectory distribution that is proportional to the route-wise rewards for a given VRP instance $\mathbf{X}$: $P(\tau|\mathbf{X}) \propto R(\tau, \mathbf{X})$.

This approach encourages the model to align with the log-scaled rewards of routes, decreasing the likelihood of getting trapped in local optima while promoting the exploration of routes with slightly lower rewards.

**Learning the Trajectory Probability:** Building on the previous concepts, for an observed training sample $(\tau, \mathbf{X}, R(\tau, \mathbf{X}))$, we can derive the trajectory balance (TB) objective from Eq. (2):

$$\mathcal{L}_{\text{ss}}(\tau;\theta) = \left( \log \frac{Z_\theta(\mathbf{X}) \prod_{t=1}^{N} P_{F_\theta}(s_{t+1}|s_t)}{R(\tau, \mathbf{X})} \right)^2 \tag{6}$$

The learnable parameters include the initial flow estimator $Z_\theta$ and the forward probability function $P_{F_\theta}$. It is important to note that the backward probability is constant, given by $P_B(s_{t-1}|s_t) = 1$, since route generation follows a tree structure rather than a directed acyclic graph, meaning each node in the tree has only one parent.

## 4.4 Problem Symmetry $\mathcal{L}_{\text{ps}}$

As defined in Definition 1 and Proposition 1, the symmetrical geometric transformation have the same generation tree graph. Thus we can obtain the relationship between the initial state flow of problem $\mathbf{X}$ and the symmetric transformation $M(\mathbf{X})$ as follows.

**Proposition 3.** *If $\mathbf{X}$ and $M^\alpha(\mathbf{X})$ are Problem Symmetry, the initial state flow of $\mathbf{X}$ is proportional to the initial state flow of $M(\mathbf{X})$, i.e., $Z(\mathbf{X}) = \frac{1}{\alpha} Z(M^\alpha(\mathbf{X}))$.*

We introduce a bias penalty loss for penalizing the deviation between the initial state flow prediction $Z(\mathbf{X})$ for problem $\mathbf{X}$ and the initial state flow prediction $Z(M^\alpha(\mathbf{X}))$ for the symmetric problem $M^\alpha(\mathbf{X})$, which is equipped with problem symmetry.

$$\mathcal{L}_{\text{ps}} = \mathbb{E}_{M \sim \mathcal{M}} \Big( Z_\theta(\mathbf{X}) - \frac{1}{\alpha} Z_\theta(T^\alpha(\mathbf{X})\Big)^2 \tag{7}$$

where $\mathcal{M}$ represents the distribution of random symmetry transformation matrices, and $M$ is a symmetry transformation matrix sampled from $\mathcal{M}$.

## 5 Experiments

### 5.1 Setup

We evaluate our proposed model against several baseline methods for both the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP). These include non-learnable baselines (heuristics) such as Concorde [1], LKH3 [6], and Gurobi [12], as well as recent neural constructive baselines: AM [10], POMO [11], MDAM [18], and Sym-NCO [8]. The performance of our model is assessed against these baselines using three metrics: tour length (Obj), optimality gap (Gap), and evaluation time (Time), with lower values indicating better performance (Table 2).

**Table 1.** Comparison with various baselines on TSP.

| Method | TSP20 | | | TSP50 | | | TSP100 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Len. | Gap | Time | Len. | Gap | Time | Len. | Gap | Time |
| Concorde | 3.83 | - | 5 m | 5.69 | - | 13 m | 7.76 | - | 1 h |
| LKH3 | 3.83 | 0.00% | 42 s | 5.69 | 0.00% | 6 m | 7.76 | 0.00% | 25 m |
| Gurobi | 3.83 | 0.00% | 7 s | 5.69 | 0.00% | 2 m | 7.76 | 0.00% | 17 m |
| OR Tools | 3.86 | 0.94% | 1 m | 5.85 | 2.87% | 5 m | 8.06 | 3.86% | 23 m |
| AM(g.) | 3.84 | 0.19% | <1 s | 5.76 | 1.21% | 1 s | 8.03 | 4.53% | 2 s |
| AM(s.) | 3.83 | 0.07% | 1 m | 5.71 | 0.39% | 5 m | 7.92 | 1.98% | 22 m |
| POMO(g.) | 3.83 | 0.12% | <1 s | 5.73 | 0.64% | 1 s | 7.84 | 1.07% | 2 s |
| POMO(s.) | 3.83 | 0.04% | <1 s | 5.70 | 0.21% | 2 s | 7.80 | 0.46% | 11 s |
| MDAM(g.) | 3.83 | 0.13% | 5 s | 5.73 | 0.65% | 15 s | 7.93 | 2.19% | 36 s |
| MDAM(s.) | 3.83 | 0.04% | 2 m | 5.70 | 0.23% | 7 m | 7.80 | 0.48% | 20 m |
| Sym-NCO(g.) | 3.83 | 0.12% | <1 s | 5.72 | 0.52% | 1 s | 7.84 | 0.94% | 2 s |
| Sym-NCO(s.) | 3.83 | 0.03% | <1 s | 5.70 | 0.21% | 2 s | 7.79 | 0.39% | 13 s |
| **GSE-VRPs**(g.) | 3.83 | 0.11% | <1 s | 5.71 | 0.49% | 1 s | 7.84 | 0.91% | 2 s |
| **GSE-VRPs**(s.) | 3.83 | 0.03% | <1 s | 5.70 | 0.12% | 2 s | 7.77 | 0.21% | 11 s |

### 5.2 Results

Table 1 presents a comparison of GSE-VRPs's performance on the TSP against several baselines. The first group of baselines comprises results from Concorde, along with other significant non-learning-based heuristic methods. We obtained the optimal solutions by running Concorde independently, while the data for the other solvers were sourced from Wu et al. [17] and Kool et al. [10]. The second group of baselines consists of deep reinforcement learning (RL) construction-based approaches documented in the literature [8,11,18]. In the third group,

**Table 2.** Comparison with various baselines on CVRP.

| Method | CVRP20 | | | CVRP50 | | | CVRP100 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Len. | Gap | Time | Len. | Gap | Time | Len. | Gap | Time |
| LKH3 | 6.12 | - | 2 h | 10.38 | - | 7 h | 15.68 | - | 12 h |
| Gurobi | 6.42 | 4.48% | 2 m | 11.22 | 8.12% | 12 m | 17.14 | 9.34% | 1 h |
| AM(g.) | 6.40 | 4.45% | <1 s | 10.93 | 5.34% | 1 s | 16.73 | 6.72% | 3 s |
| AM(s.) | 6.24 | 1.97% | 3 m | 10.59 | 2.11% | 7 m | 16.16 | 3.09% | 30 m |
| POMO(g.) | 6.35 | 3.72% | <1 s | 10.74 | 3.52% | 1 s | 16.15 | 3.00% | 3 s |
| POMO(s.) | 6.17 | 0.82% | 1 s | 10.49 | 1.14% | 4 s | 15.83 | 0.98% | 19 s |
| MDAM(g.) | 6.30 | 2.86% | 7 s | 10.74 | 3.39% | 16 s | 16.40 | 4.86% | 45 s |
| MDAM(s.) | 6.14 | 0.40% | 3 m | 10.50 | 1.19% | 9 m | 16.03 | 2.49% | 1h |
| Sym-NCO(g.) | 6.30 | 2.93% | <1 s | 10.75 | 3.48% | <1 s | 16.10 | 2.88% | 2 s |
| Sym-NCO(s.) | 6.22 | 1.69% | 1 s | 10.48 | 1.00% | 6 s | 15.87 | 1.46% | 16 s |
| **GSE-VRPs**(g.) | 6.30 | 2.85% | <1 s | 10.74 | 3.38% | 1 s | 16.01 | 2.09% | 3 s |
| **GSE-VRPs**(s.) | 6.14 | 0.41% | 1 s | 10.47 | 0.91% | 7 s | 15.75 | 0.48% | 18 s |

we present results from the Attention Model (AM), which was trained using our implementation of the generative flow network that incorporates symmetry enhancement.

For 10,000 random instances of TSP20 and TSP50, GSE-VRPs finds near-optimal solutions with optimality gaps of 0.03% in just a few seconds and 0.17% in tens of seconds, respectively. For TSP100, GSE-VRPs achieves an optimality gap of 0.21% within one minute, significantly surpassing all other learning-based heuristics in terms of both solution quality and computation time. For CVRP, our results closely match the ground-truth solutions, yielding average optimality gaps of 0.41%, 0.91%, and 0.48% on instances with $n = 20$, $n = 50$, and $n = 100$, respectively. The total runtime of our method remains competitive compared to all other learning-based baselines.

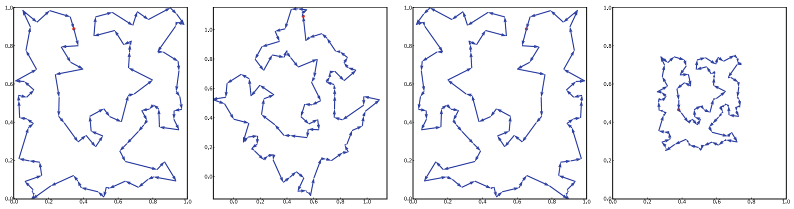


**Fig. 3.** Example TSP trajectories given by GSE-VRPs.

Figure 3 and 4 shows some trajectories obtained from symmetric problem instances on TSP100 and CVRP100 respectively. (a) is the original problem,

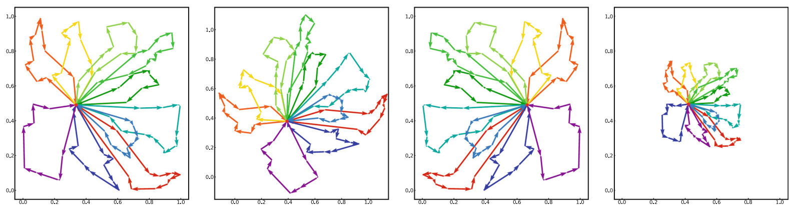


**Fig. 4.** Example CVRP trajectories given by GSE-VRPs.

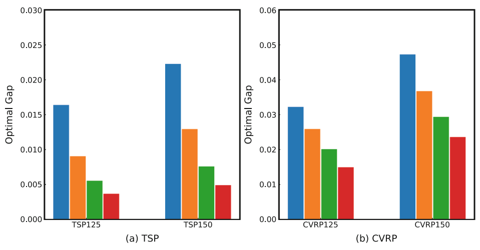


**Fig. 5.** Generalization test.

and (b), (c), and (d) are the rotation, reflection, and 0.5x uniform scaling of (a), respectively. It can be shown that although each symmetric instance has a different starting point and trajectory, it forms the same Hamiltonian circuit for TSP. Compared to the TSP, the GSE-VRPs outputs the same trajectory for different symmetric instances of the CVRP, except for instances of uniform scaling transformations.

Although all baselines were trained on $n = 100$, they struggled to generalize effectively to other problem sizes, particularly the AM model. This led to significantly poorer performance, with optimality gaps ranging from 2.5% to 4%, compared to the i.i.d. test results reported in their original papers. For example, POMO achieves an optimality gap of only 0.46% compared to LKH3 on TSP100 when both training and testing are conducted on $n = 100$. However, in our cross-size experiments, this gap increased to 0.98%. In contrast, GSE-VRPs demonstrated significantly improved generalization. For instance, GSE-VRPs reduced the gap by 1.69% (0.19% vs. 1.88%) compared to AM. Figure 5(b) shows that cross-size generalization is also challenging for neural baselines, with gaps ranging from 1.97% to 4.51%. In contrast, our method consistently produces high-quality solutions with shorter tour lengths and demonstrates greater robustness to size variations, achieving gaps as low as 1.47%. This enhancement underscores the effectiveness of our approach in utilizing the inherent symmetry of the problem.

## 6 Conclusion

Our research addresses the challenges of tackling VRPs using unsupervised learning methodologies. We contribute to this field by introducing the principled GSE-VRPs decision-making framework for VRPs. By integrating probabilistic inference with sequential decision-making, GSE-VRPs represent a promising approach to discovering a diverse array of high-quality candidate solutions in VRPs. Through extensive numerical experiments, we demonstrate the effectiveness and efficiency of GSE-VRPs in solving NP-hard VRPs. Our results highlight the ability of GSE-VRPs to generate a diverse range of high-quality solutions.

## References

1. Applegate, D., Bixby, R., Chvatal, V., Cook, W.: Concorde TSP solver (2006)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 (2016)
3. Bengio, E., Jain, M., Korablyov, M., Precup, D., Bengio, Y.: Flow network based generative models for non-iterative diverse candidate generation. Adv. Neural. Inf. Process. Syst. **34**, 27381–27394 (2021)
4. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021)
5. Duan, L., et al.: Efficiently solving the practical vehicle routing problem: a novel joint learning approach. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 3054–3063 (2020)
6. Helsgaun, K.: An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Roskilde Roskilde Univ. **12**, 966–980 (2017)
7. Kim, M., Park, J., et al.: Learning collaborative policies to solve np-hard routing problems. Adv. Neural. Inf. Process. Syst. **34**, 10418–10430 (2021)
8. Kim, M., Park, J., Park, J.: Sym-nco: leveraging symmetricity for neural combinatorial optimization. Adv. Neural. Inf. Process. Syst. **35**, 1936–1949 (2022)
9. Konstantakopoulos, G.D., Gayialis, S.P., Kechagias, E.P.: Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification. Oper. Res. Int. J. **22**(3), 2033–2062 (2022)
10. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475 (2018)
11. Kwon, Y.D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: Pomo: policy optimization with multiple optima for reinforcement learning. Adv. Neural. Inf. Process. Syst. **33**, 21188–21198 (2020)
12. LLC Gurobi Optimization: Gurobi Optimizer Reference Manual (2021)
13. Ma, Y., et al.: Learning to iteratively solve routing problems with dual-aspect collaborative transformer. Adv. Neural. Inf. Process. Syst. **34**, 11096–11107 (2021)
14. Malkin, N., Jain, M., Bengio, E., Sun, C., Bengio, Y.: Trajectory balance: improved credit assignment in gflownets. Adv. Neural. Inf. Process. Syst. **35**, 5955–5967 (2022)
15. Qin, Z., et al.: Ride-hailing order dispatching at didi via reinforcement learning. INFORMS J. Appl. Anal. **50**(5), 272–286 (2020)
16. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Advances in Neural Information Processing Systems, vol. 28 (2015)

17. Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A.: Learning improvement heuristics for solving routing problems. IEEE Trans. Neural Netw. Learn. Syst. **33**(9), 5057–5069 (2021)
18. Xin, L., Song, W., Cao, Z., Zhang, J.: Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 12042–12049 (2021)

# Edge-Unfolding Polycubes with Orthogonally Convex Layers

Mirela Damian[1](✉) and Henk Meijer[2]

[1] Villanova University, Villanova, USA
mirela.damian@villanova.edu

[2] University College Roosevelt, Middelburg, The Netherlands

**Abstract.** A *polycube* is an orthogonal polyhedron composed of unit cubes glued together along entire faces, homeomorphic to a sphere. A polycube *layer* is the section of the polycube that lies between two horizontal cross-sections of the polycube at unit distance from each other. An *edge unfolding* of a polycube involves cutting its surface along any of the constituent cube edges and flattening it into a single, non-overlapping planar piece. We show that any polycube with orthogonally convex layers can be edge unfolded.

**Keywords:** Unfolding · polycube · orthostack · orthogonally convex

## 1 Introduction

An *unfolding* of a polyhedron involves cutting its surface and flattening it into a single, non-overlapping planar piece called an *unfolding net*. *Edge unfoldings* limit cuts to the edges of the polyhedron, whereas *general unfoldings* allow cuts anywhere, including across face interiors.

The problem of unfolding polyhedra has been long studied. For non-convex polyhedra, it has been shown that edge cuts alone are not sufficient to guarantee an unfolding [1,2], and it remains unknown whether all non-convex polyhedra have a general unfolding. In contrast, all convex polyhedra have a general unfolding [13, Ch. 22], but it remains unknown whether all convex polyhedra have an edge unfolding.

Research on unfolding non-convex objects has predominantly focused on orthogonal polyhedra, a class of polyhedra whose edges and faces meet at right angles. Since edge unfoldings are not always possible for orthogonal polyhedra [2], unfolding algorithms typically employ additional non-edge cuts, which are limited to a refined grid structure on the surface. The surface is initially subdivided into rectangular grid faces by introducing new edges at the intersections of the surface with axis-perpendicular planes passing through each vertex. This subdivision can be further refined by dividing each grid face into a finer grid

Preliminary ideas explored in this work were first introduced in [11].

of $a \times b$ orthogonal subfaces, for positive integers $a, b \geq 1$. Cuts are then permitted along any of these newly introduced grid lines, allowing for more flexible unfolding strategies.

Progressive results have reduced the amount of refinement required for unfolding orthogonal polyhedra homeomorphic to a sphere. The initial method [10] used exponential refinement, later improved to quadratic [5], then linear [4]. These techniques were further extended to unfold genus-2 orthogonal polyhedra using linear refinement [6]. Constant refinement has only been achieved for specialized classes of orthogonal polyhedra. These include orthostacks using $1 \times 2$ refinement [2], Manhattan Towers using $4 \times 5$ refinement [9], and polycube trees using $4 \times 4$ refinement [7]. Unfolding techniques with cuts restricted to grid edges exist for orthotubes [2,14], well-separated orthotrees [8], orthostacks with rectangular faces [3], and orthostacks with rectangular slabs [18]. Also, single-layer polycubes with sparse cubic holes [16] and general cubic holes [17] can be unfolded with cuts restricted to the cube edges.

This paper explores edge unfoldings of polycubes (problem 64 from The Open Problems Project [15]). A *polycube* is an orthogonal polyhedron composed of unit cubes[1] glued together along entire faces, homeomorphic to a sphere. In the case of polycubes, all edges of the constituent cubes are available for cuts. This paper considers polycubes with *orthogonally convex layers*, where each layer intersects a line parallel to a coordinate axis in either a single line segment or not at all. We show that *any polycube with orthogonally convex layers has an edge unfolding.*

## 2 Terminology

Throughout this paper, $\mathcal{O}$ refers to a polycube with orthogonally convex layers. We use the term *face* to refer to a face of $\mathcal{O}$ and the term *cell* to specifically mean one of the individual unit cube faces that make up the larger faces. Two non-overlapping surface pieces of $\mathcal{O}$ are considered *adjacent* if their boundaries share at least one cell edge.

In space we use the term *vertical* to refer to the $z$-direction, and the term *horizontal* to refer to the $x$- and $y$-directions. We categorize faces based on the direction of their outward normals: right is $+x$; left is $-x$; front is $+y$; back is $-y$; top is $+z$; and bottom is $-z$. Let $z_0, z_1, \ldots, z_m$ be the distinct $z$-coordinates of the vertices of $\mathcal{O}$. Let $z = z_i$ denote the *i-plane*. We define the *i-band* to be the collection of vertical cells parallel to the $z$-axis that lie between the $(i-1)$-plane and the $i$-plane, for $i = 1, 2, \ldots, m$. Two $i$-band cells $a$ and $b$ are considered *opposite* if they are parallel to each other and enclosed by two planes orthogonal to $a$, passing through the vertical sides of $a$. By the definition of a polycube, each $i$-band is the boundary of an extruded simple orthogonal polygon and is therefore connected. The layer $\mathcal{O}_i$ is the portion of $\mathcal{O}$ bounded by the $i$-band, the $(i-1)$-plane and the $i$-plane, which forms a prism with an orthogonal polygon as its base. Thus, $\mathcal{O}$ can be viewed as a collection of orthogonally convex layers

[1] If rectangular boxes are used in place of unit cubes, this is known as an *orthostack*.

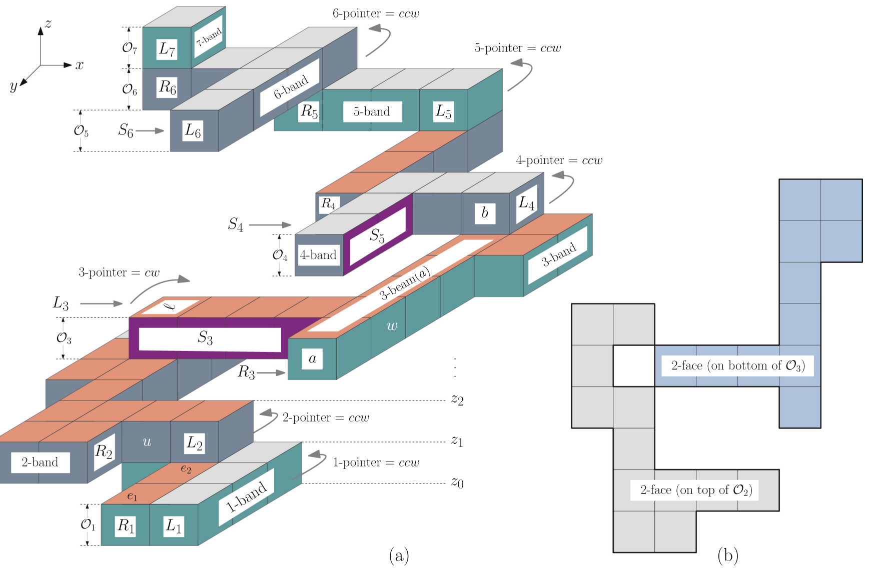


**Fig. 1.** (a) Polycube $\mathcal{O}$ with 7 orthogonally convex layers; visited $i$-band segments are delimited by $L_i$ and $R_i$, and $i$-bridges are highlighted in brown, for $i = 1, \ldots, 6$. ($S$-labels will be discussed in Sect. 4.4.) (b) Top and bottom $i$-faces of $\mathcal{O}$, for $i = 2$.

$\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_m$ stacked in this order along the positive $z$-axis. See Fig. 1a for an example of a polycube with 7 orthogonally convex layers.

An *$i$-face* is a face of $\mathcal{O}$ that lies in the $i$-plane. If located on top of $\mathcal{O}_i$, $f$ is referred to as a *top* face. Otherwise, $f$ is on the bottom of $\mathcal{O}_{i+1}$ and is referred to a *bottom* face. Refer to Fig. 1b. Similar definitions apply to cells.

For any straight band segment $S$ adjacent to an $i$-face along an edge segment $e$, *$i$-beam*$(S)$ is defined as the $i$-face piece illuminated by rays emitted from $e$ in the direction orthogonal to $S$. When talking about $i$-beam$(a)$, for a single band cell $a$, we refer to $a$ as an *anchor* to suggest the possibility of fastening (or "anchoring") the beam to $a$ during unfolding. We say that $i$-beam$(a)$ is *anchored* on $a$, or more broadly, on the band containing $a$. Note that *$i$-beam*$(a)$ has two parallel *anchors*: one is $a$, and the other is the band cell $b$ that is parallel to $a$ and adjacent to *$i$-beam*$(a)$. The cell $b$ may lie on the $i$-band, opposite to $a$, or it may be on the $(i+1)$-band. In Fig. 1a for example, 3-beam$(a)$ is anchored on cells $a$ and $b$, which are on the 3-band and the 4-band, respectively. Two beams are considered *parallel* if their anchors are parallel.

**Lemma 1. (Reformulation of Lemma 2 from [3]to match our terminology).** *The perimeter of any $i$-face is partitioned into two contiguous components, one adjacent to the $i$-band and the other adjacent to the $(i+1)$-band. Additionally, the boundary of any $i$-face contains two opposite cell edges, one on the $i$-band and the other on the $(i+1)$-band.*

## 3 Band Segments and Bridges

The main idea of our unfolding algorithm is simple. For each $i$-band, we identify two band cells, $L_i$ and $R_i$ (representing the leftmost and rightmost band cells in the unfolding net), and an unfolding direction *$i$-pointer* $\in \{cw, ccw\}$ to guide the unfolding. Here, $cw$ and $ccw$ stand for clockwise and counterclockwise, respectively, as viewed from $+z$.

Consider any two $i$-band cells, $a$ and $b$. We use *$i$-band*$[a, b]$ to denote the *closed* $i$-band segment that stretches from $a$ to $b$ inclusively, in the direction of the $i$-pointer. Similarly, *$i$-band*$(a, b)$ represents the *open* $i$-band segment, excluding $a$ and $b$. By these definitions, the $i$-band is the union of $i$-band$[a, b]$ and $i$-band$(b, a)$.

The unfolding algorithm cuts along the perimeter of each $i$-band$[L_i, R_i]$ and unfolds it as a horizontal rectangular piece in the plane, with $L_i$ to the left and $R_i$ to the right. Band segments $i$-band$[L_i, R_i]$ and $i$-band$[L_{i+1}, R_{i+1}]$ of consecutive layers are selected such that $R_i$ and $L_{i+1}$ are parallel, and $i$-beam$(R_i)$ lies left of (or coincides with) $i$-beam$(L_{i+1})$ in the unfolding. The $i$-beams delimited by (and including) $i$-beam$(R_i)$ and $i$-beam$(L_{i+1})$ form the a connected component connecting the two band segments, which we refer to as the *$i$-bridge*. The $i$-band cells outside of $i$-band$[R_i, L_i]$ are handled separately. With few exceptions, $i$-beams are attached above and below appropriate anchors in the band unfoldings.

We introduce a few more definitions. For any $i$-band cell $a$, if $a$ is part of $i$-band$[L_i, R_i]$, then $a$ is called *visited*; otherwise, $a$ is *unvisited*.

For a fixed $i$-band cell $r$, $i$-clip$(r)$ is defined as follows. If $i$-beam$(r)$ is empty, then $i$-clip$(r)$ is also empty. Otherwise, $i$-clip$(r)$ is the portion of the $i$-face delimited by (and including) $i$-beam$(r)$ and extending in the direction of the $i$-pointer from $r$.

This definition is depicted in Fig. 2. Note that $i$-clip$(r)$ is connected.

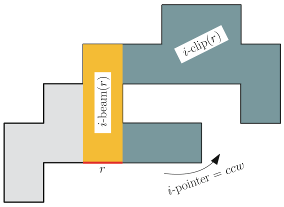


**Fig. 2.** Top view of an $i$-face; $i$-clip$(r)$ is the dark-shaded piece.

### 3.1 Selecting Band Segments and Bridges

In this section, we show how to select $L_i$, $R_i$, the $i$-bridge, and the $i$-pointer, for increasing $i \geq 1$. We start by setting 1-pointer $= ccw$. Let $F$ be an arbitrary top 1-face of $\mathcal{O}_1$ and let $e_1$ and $e_2$ be two opposite grid edges of $F$ adjacent to the 1-band and the 2-band, respectively. By Lemma 1, $e_1$ and $e_2$ exist. We set $R_1$ to be the 1-band cell adjacent to $e_1$, and $L_1$ to be the 1-band cell adjacent to $R_1$ in the direction of the 1-pointer (i.e., $ccw$). The 1-bridge is selected to be 1-beam$(R_1)$. We also set $L_2$ to be the 2-band cell adjacent to $e_2$, and set the 2-pointer to $ccw$. Refer to Fig. 1a.

For $i > 1$, we assume that both $L_i$ and the $i$-pointer for traversing the $i$-band are fixed. This assumption holds as we handle $i$-bands sequentially, for increasing $i$. We now show how the $(i + 1)$-pointer, $R_i$, $L_{i+1}$, and the *$i$-bridge* connecting the two are selected by our unfolding approach.

**Definition 1 ($R_i$, $L_{i+1}$, $i$-bridge, $(i{+}1)$-pointer).** *Starting at $L_i$, walk around the i-band in the direction of the i-pointer. Define $R_i$ to be the i-band cell* last *encountered along this walk, such that i-clip($R_i$) is either empty or is adjacent to an $(i+1)$-band cell parallel to $R_i$.*
*If i-clip($R_i$) is empty, then $L_{i+1}$ is the $(i+1)$-band cell adjacent to $R_i$, the i-bridge is empty, and the $(i+1)$-pointer is the same as the i-pointer. Otherwise, imagine i-clip($R_i$) partitioned into beams parallel to i-beam($R_i$):*

*(a) Define $L_{i+1}$ to be the $(i+1)$-band cell parallel to $R_i$ and adjacent to i-clip($R_i$), that minimizes the number of beams delimited by i-beam($R_i$) and i-beam($L_{i+1}$). If mutiple such cells exist, select the one that minimizes the Manhattan distance to $R_i$.*
*(b) The i-bridge comprises i-beam($R_i$), i-beam($L_{i+1}$), and the in-between beams (if any).*
*(c) If $R_i$ and $L_{i+1}$ have the same normal, then the $(i+1)$-pointer is the same as the i-pointer; otherwise, the $(i+1)$-pointer is opposite to the i-pointer.*

In Fig. 1a for instance, all bridges are marked in brown. The following lemmas establish some structural properties used to guide the unfolding process. Due to space limitations, we direct the reader to [12] for the proofs.

**Lemma 2.** *$R_i$ and $L_{i+1}$ exist and are uniquely defined.*

**Lemma 3.** *Any beam on the surface of $\mathcal{O}_i$ has at least one anchor on the i-band.*

**Lemma 4.** *For every unvisited i-band cell r:*

*(a) i-clip(r) is non-empty and lies on top of $O_i$, and*
*(b) i-clip(r) is not adjacent to an $(i+1)$-band cell parallel to r*

*Similarly, (a) and (b) hold for i-beam(r).*

**Lemma 5.** *If the i-bridge is on a bottom face, it consists of a single beam.*

**Lemma 6.** *For every unvisited i-band cell u parallel to $R_i$, i-beam(u) is non-empty and is anchored on a visited i-band cell.*

**Lemma 7.** *All i-band cells with normals opposite to that of $R_i$ are visited. Furthermore $L_i$ may not lie opposite to $R_i$.*

## 4 Unfolding Algorithm

The unfolding algorithm is fairly simple, but proving its correctness requires complex reasoning. In this paper, we present the algorithm itself. Its proof of correctness can be found in [12].
Our unfolding procedure for $\mathcal{O}$ had four stages:

**Stage 1:** Unfolding visited band segments and bridges (Sect. 4.1)
**Stage 2:** Unfolding the top surface of $\mathcal{O}$ (Sect. 4.2)
**Stage 3:** Unfolding the bottom surface of $\mathcal{O}$ (Sect. 4.3)
**Stage 4:** Unfolding remaining band pieces of $\mathcal{O}$ (Sect. 4.4)

We say that $L_i$ and $R_i$ are *quasi-adjacent* if they are orthogonal and connected by a *straight* $i$-band segment. In Fig. 1a for example, $L_2$ and $R_2$ are quasi-adjacent, whereas $L_3$ and $R_3$ are not, as they are parallel.

## 4.1 Stage 1: Unfolding Visited Band Segments and Bridges

For each $i \geq 0$, we cut out the entire visited segment $i$-band$[L_i, R_i]$ and unfold it horizontally in the plane, with $L_i$ to the left and $R_i$ to the right. The $i$-bridge, delimited by $i$-beam$(R_i)$ and $i$-beam$(L_{i+1})$ (cf. Definition 1b), unfolds as a single piece connecting $i$-band$[L_i, R_i]$ and $(i+1)$-band$[L_{i+1}, R_{i+1}]$. Refer to Fig. 7.

## 4.2 Stage 2: Unfolding the Top Surface of $\mathcal{O}$

Here we discuss the unfolding process for the top surface of $\mathcal{O}_i$, for each $i$. Assume, without loss of generality, that *i-pointer* = *ccw* (the *cw* case is symmetric).

*Unfolding Process.* We partition the top faces of $\mathcal{O}_i$ into beams parallel to $i$-beam$(R_i)$. Consider a beam $\beta$ that is not part of the $i$-bridge. By Lemma 6, at least one of $\beta$'s anchors, say $r$, is visited and on the $i$-band. If $\beta$'s other anchor $q \neq r$ is not a visited $i$-band cell, we attach $\beta$ above $r$ in the unfolding net along the shared side. If $q$ is an unvisited $i$-band cell, we also attach $q$ above $\beta$ in the unfolding net. If both of $\beta$'s anchors are visited $i$-band cells, and one of them is $L_i$, we attach $\beta$ above $L_i$. Otherwise, we attach $\beta$ above the *second* visited anchor along their shared side. This strategy minimizes the need to relocate beams later during the unfolding process.

At this point, the unfolding net includes the entire top surface of $\mathcal{O}_i$ and all $i$-band cells parallel to $R_i$. Refer to Fig. 7 for an example.

## 4.3 Stage 3: Unfolding the Bottom Surface of $\mathcal{O}$

Here we discuss the unfolding process for the bottom surface of $\mathcal{O}_i$, for each $i$. As before, we assume that *i-pointer* = *ccw* (the *cw* case is symmetric).

*Unfolding Process.* We start by partitioning the bottom faces of $\mathcal{O}_i$ as follows. If $R_i$ and $L_i$ are quasi-adjacent, we partition the bottom faces of $\mathcal{O}_i$ into beams parallel to $(i-1)$-beam$(L_i)$. Otherwise, we partition the bottom faces of $\mathcal{O}_i$ into beams parallel to $(i-1)$-beam$(R_i)$. This strategy avoids situations where a bottom beam in the partition is anchored on two unvisited $i$-band rectangles orthogonal to $R_i$.

Consider a beam $\beta$ in this partition that is not part of the $(i-1)$-bridge. By Lemma 3, at least one of $\beta$'s anchors is on the $i$-band. We determine where to attach $\beta$ to the unfolding net.

*Unfolding the Bottom of $\mathcal{O}_i$ When $R_i$ and $L_i$ are Quasi-adjacent.* In this case, $\beta$ is parallel to $(i-1)$-beam$(L_i)$. By Lemma 3, at least one of $\beta$'s anchors is on the $i$-band. By the quasi-adjacency property of $L_i$ and $R_i$, if both of $\beta$'s anchors are on the $i$-band, then at least one of them is visited. We attach $\beta$ to the unfolding net as follows:

– If both of $\beta$'s anchors are on the $i$-band, but only one of them – say, $r$ – is visited, we attach $\beta$ below $r$, and $\beta$'s unvisited anchor below $\beta$.
– If both of $\beta$'s anchors are on the $i$-band and both are visited, we attach $\beta$ below the *second* visited anchor (to minimize the potential need to relocate $\beta$ later in the unfolding process).

- Otherwise, $\beta$ has one unvisited anchor $r$ on the $i$-band, and its second anchor $q$ on the $(i-1)$-band. By Lemma 4a, $q$ is visited and therefore part of the unfolding net. In this case, we attach $\beta$ to $q$, and $r$ to $\beta$.

Furthermore, for each unvisited $i$-band cell $u$ (which must be parallel to $L_i$) adjacent to an $(i-1)$-band cell $v$ (which, by Lemma 4, must be visited), we attach $u$ to $v$ in the unfolding net. See, for example, the 2-band cell labeled $u$ in Fig. 7, attached to the 1-band. At this point, the unfolding net includes the entire surface of $O_i$.

*Unfolding the Bottom of $\mathcal{O}_i$ when $R_i$ and $L_i$ are not Quasi-Adjacent.* In this case, $\beta$ is parallel to $(i-1)$-beam$(R_i)$, which runs parallel to the beams in the top partition of $\mathcal{O}_i$. Recall that the $(i-1)$-bridge comprises beams parallel to $(i-1)$-beam$(L_i)$, so if $L_i$ is orthogonal to $R_i$, then $\beta$ may cross the $(i-1)$-bridge.

Assume first that $\beta$ does not cross the $(i-1)$-bridge. Recall that, at this point, any unvisited $i$-band cell $r$ parallel to $R_i$ sits above $i$-beam$(r)$ in the unfolding net (see discussion in Sect. 4.2). We attach $\beta$ to the unfolding net as follows:

- If $\beta$ is anchored on an unvisited $i$-band cell $r$, then $r$ is parallel to $R_i$ and is positioned on top of $i$-beam$(r)$ in the unfolding net (see Sect. 4.2). In this case we attach $\beta$ above $r$ in the unfolding net, along the shared side.
- Otherwise, if only one of $\beta$'s anchors $r$ is on the $i$-band (which must be visited, by case assumption), we attach $\beta$ below $r$.
- Otherwise, both of $\beta$'s anchors are visited $i$-band cells, in which case we attach $\beta$ below the *second* visited $i$-band anchor $r$.

Assume now that $\beta$ crosses the $(i-1)$-bridge. Refer to Fig. 3. By Lemma 5, the $(i-1)$-bridge consists of a single strip (since it lies on a bottom face) and its intersection with $\beta$ is a single square piece. Let $\beta_1$ and $\beta_2$ be the two sub-beams obtained after removing from $\beta$ the shared $(i-1)$-bridge section. If both sub-beams are empty, then $\beta$ is entirely within the $(i-1)$-bridge and is already part of the unfolding net. Otherwise, pick a non-empty sub-beam, say $\beta_1$.

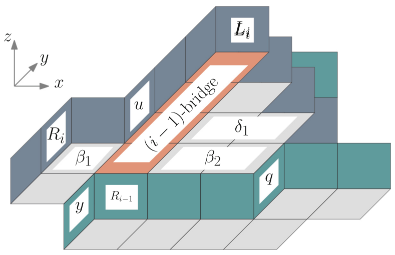


**Fig. 3.** $\beta = (i-1)$-beam$(R_i)$ crosses the $(i-1)$-bridge.

- If $\beta_1$ is anchored on an $i$-band cell $r$ (which may be visited or unvisited), then $r$ must be parallel to $R_i$ and thus already present in the unfolding net (see Sect. 4.2). In this case we attach $\beta_1$ to $r$ in the unfolding net (above $r$ if unvisited, and below $r$ if visited). For example, in the scenario depicted in Fig. 3, $\beta_1$ would get attached below $R_i$.
- Otherwise, $\beta_1$ must be anchored on an $(i-1)$-band cell $q$. Since $\beta$ crosses the $(i-1)$-bridge, it is orthogonal to $(i-1)$-beam$(R_{i-1})$, so $q$ is orthogonal to $R_{i-1}$. By Lemma 4, $q$ is visited. In this case we attach $\beta_1$ to $q$ in the unfolding net, along the shared side.

The sub-beam $\beta_2$, if non-empty, is handled similarly.

## 4.4 Stage 4: Unfolding the Remaining Band Pieces of $\mathcal{O}$

At this stage, the unfolding net includes the top and bottom surface of $\mathcal{O}$, along with all visited band cells and all $i$-band cells parallel to $R_i$, for each $i$. Moreover, if $L_i$ and $R_i$ are quasi-adjacent, the net includes the entire surface of $\mathcal{O}_i$.

For the cases where $L_i$ and $R_i$ are not quasi-adjacent, we complete the unfolding of $\mathcal{O}_i$ incrementally, for increasing $i$. Since $R_1$ and $L_1$ are adjacent, the entire surface of $\mathcal{O}_1$ is part of the unfolding net (as discussed in Sect. 3.1).

Fix $i > 1$ and assume that the entire surface of $\mathcal{O}_1, \ldots, \mathcal{O}_{i-1}$ has been incorporated in the unfolding net. Assume without loss of generality that $\mathcal{O}$ is oriented such that $R_i$ is a *front* cell (with normal $+y$) and *i-pointer* = *ccw* (the case *i-pointer* = *cw* is symmetric). By Lemma 7, all $i$-band back cells are visited, so the only surface pieces of $\mathcal{O}_i$ left to unfold are the unvisited left and right $i$-band cells, if any. If no such cells exist, the unfolding process for $\mathcal{O}_i$ is finished.

Otherwise, consider a *straight* band segment $S = i\text{-band}(a, b)$ containing unvisited left or right cells, where both $a$ and $b$ are parallel to $R_i$. Let $S^* \subseteq S$ denote the unvisited portion of $S$. To unfold $S^*$, we distinguish four cases: (1) $a$ unvisited, $b$ unvisited, (2) $a$ visited, $b$ unvisited, (3) $a$ visited, $b$ visited, and (4) $a$ unvisited, $b$ visited. Next we discuss each of these four cases in turn. Let $a'$ and $b'$ denote the $i$-band cells opposite $a$ and $b$, respectively.

**Case 1: *a* Unvisited, *b* Unvisited.** In this case $S^* = S$. By Lemma 7, $a$ and $b$ must be front cells (since $R_i$ is a front cell, by assumption), and the opposite back cells $a'$ and $b'$ are both visited.

*Unfolding Process.* Let $S' = i\text{-band}(b', a')$. Note that $S'$ is a straight band segment orthogonal to $R_i$. Since $a'$ and $b'$ are both visited, $S'$ is entirely visited. Since $a$ is unvisited, $i\text{-beam}(a) \equiv i\text{-beam}(a')$ lies on top of $O_i$ (cf. Lemma 4). Similarly, $i\text{-beam}(b) \equiv i\text{-beam}(b')$ lies on top of $O_i$. This, along with our assumption that the surface of $\mathcal{O}$ is a 2-manifold, implies that $i\text{-beam}(S')$ is non-empty and lies on the top of $\mathcal{O}_i$.

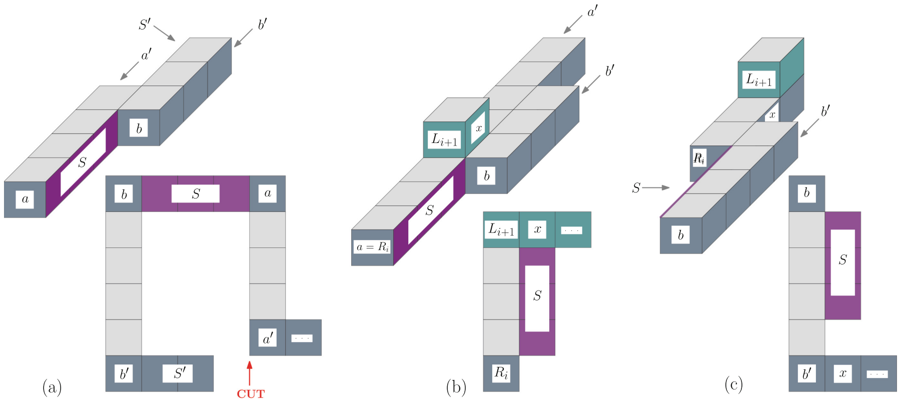


**Fig. 4.** (a) Case 1, $S$ longer than $S'$ (b) Case 2, $S$ facing *right* (c) Case 2, $S$ facing *left*.

Recall that at this point $a$ and $b$ are already in the unfolding net above $i$-beam($a'$) and $i$-beam($b'$), respectively (see Sect. 4.2). If $S$ is no longer than $S'$, we attach $S$ to $a$ (it could also be attached to $b$). Otherwise, we first cut the unfolding net along the left side of $a'$. (The cut could be along any vertical edge interior to $S'$, but for definiteness, we opt for the left edge of $a'$.) This cut disconnects the unfolding net into two components, which we then reconnect by attaching $S$ to $a$ and $b$ (so $S$ unrolls horizontally between $b$ and $a$). This process is depicted in Fig. 4a.

**Case 2: *a* Visited, *b* Unvisited.** Given that $a$ (which is parallel to $R_i$) is visited and $b$ is unvisited, it must be that $a = R_i$ is a front cell. As $b$ remains unvisited, $b$ is also a front cell (cf. Lemma 7), and $S$ excludes $L_i$. It follows that $L_i$ and $R_i$ are not quasi-adjacent, and $S^* = S$ (i.e., the entire segment $S$ is unvisited).

Since $b$ is unvisited, $i$-clip($b$) lies on top of $\mathcal{O}_i$ and is not adjacent to $L_{i+1}$ (cf. Lemma 4). Similarly, since $S$ is unvisited, $i$-beam($S$) is non-empty and lies on top of $\mathcal{O}_i$. These together imply that the $i$-bridge lies on top of $\mathcal{O}_i$ and consists of a single beam, namely $i$-beam($a$). Consequently, the $(i+1)$-pointer is identical to the $i$-pointer ($ccw$, by our assumption).

*Unfolding Process.* If $S$ is a right segment, we attach $S$ to the right side of $i$-beam($a$) in the unfolding net, along the shared side (see Fig. 4b). Otherwise, $S$ is a left segment, in which case we attach $S$ to the right side of $i$-beam($b'$) in the unfolding net, along the shared side (see Fig. 4c).

**Common Scenario.** Before delving into the specifics of the next two cases, we identify a situation shared between them. The setting for this shared scenario is as follows (refer to Fig. 5): $L_i$ is adjacent to an unvisited $i$-band cell $r$, the interior angle formed by $L_i$ with $r$ is $\pi/2$, and $S = S^*$ is the maximal straight $i$-band segment $S$ containing $r$ (orthogonal to $L_i$). The process of unfolding $S$ requires a detailed case analysis, which we address in [12] due to space limitations.

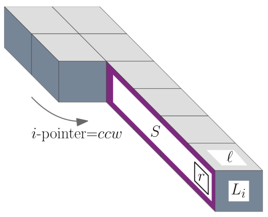


**Fig. 5.** Common scenario.

**Case 3: *a* Visited, *b* Visited.** Given that $a$ (which is parallel to $R_i$) is visited and the segment $S$ (orthogonal to $R_i$) is not entirely visited, it follows that $a = R_i$ is a front cell. Given that $b$ is visited, either $b = L_i$ or $S$ includes $L_i$. If $L_i \in S$, then $L_i$ and $R_i$ are quasi-adjacent, in which case $S$ is already part of the unfolding net (see Sect. 4.3.) Therefore, the discussion in this section is based on the assumption that $b = L_i$, in which case $S^* = S$. By Lemma 7, $L_i$ cannot lie opposite to $R_i$, therefore $L_i$ is also a front cell.

*Unfolding Process.* The case where $S$ is a left segment matches the setting for the common scenario. If $S$ is a right segment, we attach $S$ to the right side of $i$-beam($R_i$) in the unfolding net, along the shared side (see Fig. 6a).

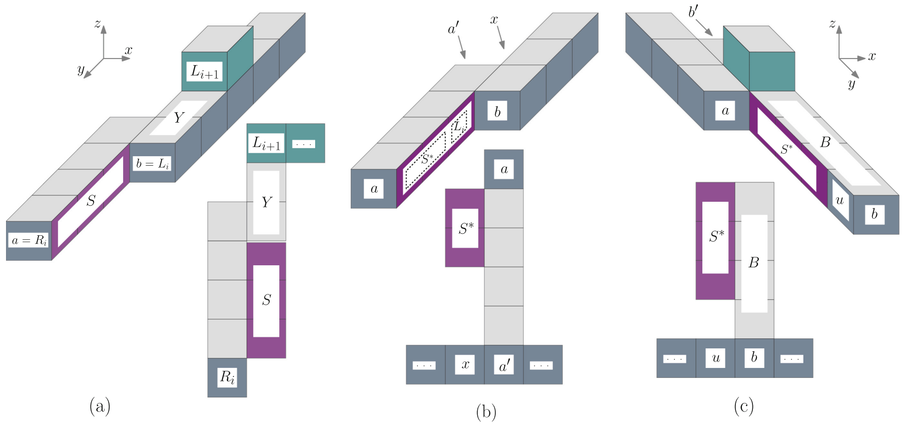


**Fig. 6.** (a) Case 3, $S$ facing *right* (b) Case 4, $S$ facing *right* (c) Case 4, $S$ facing *left*.

**Case 4: *a* Unvisited, *b* Visited.** Given that $a$ is unvisited and $R_i$ is a front cell, $a$ must also be a front cell (cf. Lemma 7). Moreover, $i$-beam($a$) lies on top of $O_i$ and is anchored on the $i$-band only (cf. Lemma 4). Since $b$ is visited, either $b = L_i$ (in which case $S^* = S$) or $S$ includes $L_i$.

*Unfolding Process.* Our unfolding procedure depends on whether $b$ is a front or a back cell, and $S$ is a left or a right segment. Assume first that $b$ is a front cell. If $S$ is a right segment, we attach $S^*$ to the left side of the $i$-beam($a'$), as shown in Fig. 6b. If $S$ is a left segment and $b = L_i$, we have the common scenario. If $S$ is a left segment and $b \neq L_i$, then $S$ includes $L_i$, so $S^* \subset S$. In this case, we will show that $S^*$ is adjacent to $B = i$-beam($b$). Our unfolding procedure first relocates $B$ to sit on top of $b$ in the unfolding net (if necessary), then attaches $S^*$ to the left side of $B$, along the shared side. See Fig. 6c.

Assume now that $b$ is a back cell. In this case $a' = b$ and $S$ is a right segment. If $L_i \in S$, we attach $S^*$ to the left side of $B$. If $L_i \notin S$, then $b = L_i$ (since $b$ is visited). This matches the setting for the common scenario.
Having exhausted all cases, the unfolding process is now complete.

## 4.5 Complete Unfolding Example

Figure 7 shows the final unfolding net for the polycube example from Fig. 1a. Observe the following:

- From a point of view facing $R_3$, the setup for $S_3$ is a horizontal mirror reflection of the setup for Case 3.
- From a point of view facing $R_4$, $S_4$ is a left segment that matches the setup for Case 2, and $S_5$ is a right segment that matches the setup for Case 1.
- From a point of view facing $R_6$, $S_6$ is a left segment that matches the setup for Case 4.

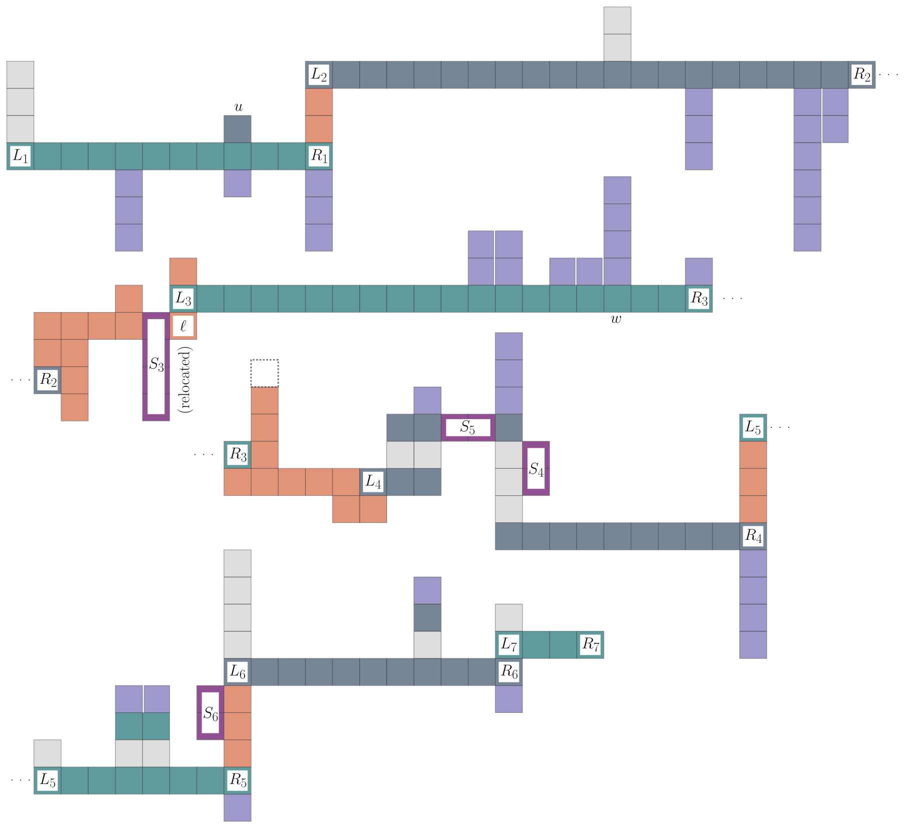


**Fig. 7.** Complete unfolding of the polycube from Fig. 1a: horizontal visited band segments are connected by orange bridges; top beams are marked in light gray, and above some of them are unvisited band cells; bottom beams are marked in light purple; and $S$-labeled segments represent unvisited band segments.

## 5 Conclusion

We show that every polycube with orthogonally convex layers can be edge unfolded. A natural extension of this work would be to remove the orthogonal convexity constraint, thereby addressing the general polycube edge unfolding problem posed in [15].

Another potential extension would be to use rectangular boxes instead of cubes as building blocks for the orthogonal polyhedron. Our unfolding algorithm works as-is for rectangular boxes, provided their height does not exceed their width and depth. Removing this height restriction would move us closer to the goal of unfolding arbitrary orthostacks, a challenge that remains open.

## References

1. Bern, M., Demaine, E., Eppstein, D., Kuo, E., Mantler, A., Snoeyink, J.: Ununfoldable polyhedra with convex faces. Comput. Geom. Theory Appl. **24**(2), 51–62 (2003)
2. Biedl, T., et al.: Unfolding some classes of orthogonal polyhedra. In: Proceedings of the 10th Canadian Conference on Computational Geometry, Montréal, Canada (1998)
3. Chambers, E.W., Sykes, K., Traub, C.M.: Unfolding rectangle-faced orthostacks. In: Proceedings of the 24th Canadian Conference on Computational Geometry, pp. 23–28 (2012)
4. Chang, Y.-J., Yen, H.-C.: Unfolding orthogonal polyhedra with linear refinement. In: Elbassioni, K., Makino, K. (eds.) ISAAC 2015. LNCS, vol. 9472, pp. 415–425. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48971-0_36
5. Damian, M., Demaine, E., Flatland, R.: Unfolding orthogonal polyhedra with quadratic refinement: the Delta-unfolding algorithm. Graphs Comb. **30**(1), 125–140 (2014)
6. Damian, M., Demaine, E., Flatland, R., O'Rourke, J.: Unfolding genus-2 orthogonal polyhedra with linear refinement. Graphs Comb. **33**(5), 1357–1379 (2017)
7. Damian, M., Flatland, R.: Unfolding polycube trees with constant refinement. Comput. Geom. **98**, 101793 (2021)
8. Damian, M., Flatland, R., Meijer, H., O'Rourke, J.: Unfolding well-separated orthotrees. In: Abstracts from the 15th Annual Fall Workshop on Computational Geometry, Philadelphia, PA (2005)
9. Damian, M., Flatland, R., O'Rourke, J.: Unfolding Manhattan towers. In: Proceedings of the 17th Canadian Conference on Computational Geometry, Windsor, Canada, pp. 211–214 (2005)
10. Damian, M., Flatland, R., O'Rourke, J.: Epsilon-unfolding orthogonal polyhedra. Graphs Comb. **23**(1), 179–194 (2007)
11. Damian, M., Meijer, H.: Edge-unfolding orthostacks with orthogonally convex slabs. In: Abstracts from the 14th Annual Fall Workshop on Computational Geometry, Cambridge, MA, pp. 20–21 (2004)
12. Damian, M., Meijer, H.: Edge-unfolding polycubes with orthogonally convex layers (2024). https://arxiv.org/abs/2407.01326
13. Demaine, E., O'Rourke, J.: Geometric Folding Algorithms: Linkages, Origami. Cambridge University Press (2007)
14. Demaine, E.D., Karntikoon, K.: Unfolding orthotubes with a dual hamiltonian path. Thai J. Math. **21**(4), 1011–1023 (2023)
15. Demaine, E.D., Mitchell, J.S.B., O'Rourke, J.: The Open Problems Project. https://topp.openproblem.net
16. Liou, M.-H., Poon, S.-H., Wei, Y.-J.: On edge-unfolding one-layer lattice polyhedra with cubic holes. In: Cai, Z., Zelikovsky, A., Bourgeois, A. (eds.) COCOON 2014. LNCS, vol. 8591, pp. 251–262. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08783-2_22
17. Minaríc, J.: Unfolding some classes of one-layer polycubes. In: Proceedings of the 34th Canadian Conference on Computational Geometry, Toronto, Canada, pp. 243–248 (2022)
18. Pernicová, K.: Grid-edge unfolding orthostacks with rectangular slabs. In: Proceedings of the 36th Canadian Conference on Computational Geometry, pp. 65–70 (2024)

# $B$-Matching Interdiction Problem on Bipartite Graphs with Unit Weight and Multi-dimensional Budgets

Ruiqing Sun(✉) and Weidong Li

School of Mathematics and Statistics, Yunnan University, Kunming, China
ruiqing2020@126.com

**Abstract.** In this paper, we consider a network optimization interdiction problem, called the $b$-matching interdiction problem on bipartite graphs with unit weight and multi-dimensional budgets. Given an undirected bipartite graph $G$, every edge of $G$ has a multi-dimensional interdiction costs and budget. The goal is to remove a subset of the edges constrained to a multi-dimensional budget, such that the maximum $b$-matching in the resulting graph is minimized. Let $d$ be the dimension of the leader's budget. We first show that $b$-matching interdiction problem is W[1]-hard with respect to the budget for the number of interdicted edges when $d = 2$ and graph contain only isolated edges. Then, we propose a $(d + 1)$-approximation algorithm on bipartite graphs via the iterative rounding method. Finally, we also show that our iterative rounding method is a 2-approximation algorithm when $d$ is a constant.

**Keywords:** Interdiction · $B$-matching · Complexity · Approximation algorithm · Bipartite graph

## 1 Introduction

In recent years, the Stackelberg game [20] theory has attracted more and more attention, which includes two types of decision makers: one leader and one follower. Firstly, the leader executes a strategy. According to the leader's strategy, the follower chooses his own strategy. Interdiction problems is a special case of Stackelberg game. In an optimization interdiction problem, leaders first employ strategies to minimize the follower's advantage and the follower next maximizes the its advantage according to the leader's strategies. The amount of interdiction problems work has been studied on shortest path interdiction [8,10], maximum spanning tree interdiction [7,23], and maximum flow interdiction [1]. For more interdiction problem results, please see the survey [19]. Our problem also included generally in the area of bilevel optimization. Jeroslow [13] showed that bilevel optimization problems are NP-hard even if when the objectives and the constraints are linear. For more results on bilevel optimization, please see the survey [2].

For results related to matching interdiction problem, Zenklusen [22] considered the edge/vertex interdiction problem, the edges/vertices can be removed from graph. The goal is to find a edge/vertex set with cost is at most a given threshold and minimize the weight of the maximum matching in residual graph (not contain removed edge set and vertex set). Zenklusen [21] proved that the edge interdiction problem is NP-complete on simple bipartite graphs with unit edge weights and unit interdiction costs. Zenklusen [22] gave a 4-approximation for the general graph when all edge weights are unit and a 2-approximation for the bipartite graph case. They also showed that vertex interdiction problem is polynomial time solvable for unweighted bipartite graphs and NP-hard for bipartite graphs when edge weights are bounded by a constant. Finally, they also proposed an FPTAS for bounded treewidth graphs. Subsequently, Dinitz et al. [3] obtained an constant-factor approximation to the matching interdiction problem without the unit weight assumption which is called packing interdiction problem. They designed a $O(\log q \cdot \min\{q, \log k\})$-approximation algorithm for packing interdiction problem, where $q$ is the row-sparsity of the packing linger programming and $k$ is the column-sparsity.

The $b$-matching problem, in addition to being an extension of the classical matching problem, it has a wide range of practical applications in semi-supervised learning, spectral clustering, graph embedding, and manifold learning [11,12,18] and other fields. For more $b$-matching algorithm, please see the papers [9,14]. Motivated by the studies in [9,22], this paper considers an optimization interdiction problem called the $b$-matching interdiction problem with unit edge weights and multi-dimensional budgets. In a $b$-matching network, the follower wants to find a matching where each vertex matches a maximum of $b$ times to maximize the total weight, while the leader interdicts the edge under budget (multi-dimensional) constraints to reduce the maximum profit of $b$-matching. Let $d$ be the dimension of the leader's budget. Our problem is clearly NP-hard, since when $b_v = 1$ and $d = 1$, our problem is matching interdiction with unit edge weight in [22]. In this paper, we show that $b$-matching interdiction problem is W[1]-hard with respect to the budget for the number of interdicted edges when $d = 2$. We also give a $(d+1)$-approximation algorithm for this problem and it is also a 2-approximation algorithm when $d$ is a constant.

The remainder of this paper is structured as follows. In Sect. 2, we provide the problem formulation of our problem. In Sect. 3, we show that $b$-matching interdiction problem is W[1]-hard with respect to the budget for the number of interdicted edges when $d = 2$. In Sect. 4, we design a $(d+1)$-approximation algorithm for edge interdiction problem by using iterative rounding technique and also show that it is a 2-approximation algorithm when $d$ is a constant. In the last section, we present conclusions and future work.

## 2 Preliminaries and Problem Formulation

Matching theory is one of the classical and most important topics in combinatorial theory and optimization. Recall that a matching is a set of pairwise disjoint edges. The maximum weight $b$-matching problem is a generalization of maximum weight matching, i.e. $b_v = 1$ for each edge $e$ and vertex $v$. In this paper, we consider $b$-matching edge interdiction problem on bipartite graphs with unit weights and multi-dimensional budgets. Let $d$ be the dimension of the leader's budget. The problem can be described as follows:

Given a bipartite graph $G = (V, E)$, an $d$-dimensional interdiction cost vector $\boldsymbol{c} = (c_e^1, \cdots, c_e^d)$ and an $d$-dimensional budget vector $\boldsymbol{B} = (B_1, \cdots, B_d)$, and a value $b_v$ for $v \in V$. The goal is to find a set $R \subseteq E$ that minimizes the maximum $b$-matching in $G \setminus R$ with cost $c^i(R) = \sum_{e \in R} c_e^i$ at most $B_i$, for $i = 1, 2, \cdots, d$. Without loss of generality, we assume that all parameters are positive integers. Given an undirected graph $G = (V, E)$ and a vertex $v \in V$, we denote $\delta(v)$ as the set of edges which exactly one endpoint is $v$. By defining two integer variables $y_e, x_e$, the problem can be formulated as follows:

$$\begin{cases} \min \ z \\ s.t. \ \sum\limits_{e \in E} c_e^i y_e \le B_i, i = 1, 2, \cdots, d \\ y_e \in \{0, 1\}, \ \forall e \in E, \\ z = \max\{\sum\limits_{e \in E} x_e(1 - y_e) \\ \sum\limits_{e \in \delta(v)} x_e \le b_v, \ \forall v \in V, \\ x_e \in \mathbb{Z}_{\ge 0}, \ \forall e \in E.\} \end{cases} \tag{1}$$

Note that the follower problem can be solved by the Edmonds-Karp algorithm [5] in polynomial time. In the rest of the paper, we are more concerned with the solution $\boldsymbol{y}$, because the solution $\boldsymbol{x}$ can always be determined by using a polynomial time algorithm to solve the follower problem on the residual graph. However, if we split each vertex $v$ into $b_v$, the execution time of the iterative rounding algorithm in [22] is polynomial in the number of vertexes, which are very large in reality. Thus, we design a new polynomial time approximation algorithm which is not dependent on the $b_v$ via the iterative rounding method.

## 3 Complexity

In this section, we show that the $b$-matching interdiction problem is W[1]-hard with respect to the budget for the number of interdicted edges on graphs consisting only of isolated edges and $d = 2$. There is a computational complexity result for the matching interdiction problem.

**Theorem 1 ([22], Theorem 1).** *The edge interdiction problem is NP-complete on graphs consisting only of isolated edges and $d = 2$ and $b_v = 1$ for each $v \in V$.*

Next, we analyze the interdiction problem from the point of parameterized complexity view. Similarly to the reductions used to show NP-hardness of some (non-parameterized) problem, the standard technique for showing parameterized hardness is through a parameterized reduction. For more parameterized complexity theory, please see the book [4].

**Definition 1.** *A parameterized reduction from a problem P2 to a problem P1 is an algorithm mapping an instance $I_2$ of $P_2$ with parameter $k_2$ to an instance $I_1$ of $P_1$ with a parameter $k_1$ in time $f(k_2)|I_2|O(1)$ such that $k_1 \leq f(k_2)$ and $I_1$ is a yes-instance for $P_1$ if and only if $I_2$ is a yes-instance for $P_2$.*

Let $k$ be the budget for the number of interdicted edge. We will show that this interdiction problem is W[1]-hard with respect to the budget of interdicted edge $k$ even if $d = 2$ and any edge interdiction cost equal to the edge weight.

**Theorem 2.** *The edge interdiction problem with respect to $k$ when $d = 2$, is W[1]-hard on graphs consisting only of isolated edges and $b_v = 1$ for each $v \in V$ for parameter $k$, even if the any edge interdiction cost equal to the edge weight.*

*Proof.* We use the $k$ - Subset Sum problem for the reduction. It is known to be W[1]-hard, parameterized by $k$ (see Fellows et al. [6]).

**$k$-Subset Sum:** Given a set of $n$ integers $S = \{a_1, a_2, \cdots, a_n\}$ and two values $B, k \in N$. The goal is to select a subset $S_1 \subseteq S$ with $k$ elements of the given integers $a_i$ that sum up to $B$.

For a given instance of the $k$ - Subset Sum, we construct an instance of the decision version of edge interdiction problem as follows. Let $R$ be a interdiction edge set.

- $|E| = n$ isolated edges, $2n$ vertexes and $E = \{1, 2, \cdots, n\}$.
- $c_i = w_i = a_i$ for $i = 1, 2, \cdots, n$.
- $|R| = k$.
- The decision version asks whether there is a interdiction edge set $R$ such that

$$\sum_{i \in R} c_i \leq B \text{ and } \sum_{i \in E \setminus R} w_i \leq \sum_{i \in [n]} a_i - B.$$

If the instance of $k$-Subset Sum is "yes", then we construct an interdiction edge set $R$ in the following way: Select $k$ edges $i$ with $a_i \in S_1$ form a $R$. It can be seen that, $\sum_{i \in R} c_i = \sum_{i \in S_1} a_i = B$. It follows that $\sum_{i \in E \setminus R} w_i = \sum_{i \in E} w_i - \sum_{i \in R} w_i = \sum_{i \in E} c_i - \sum_{i \in R} c_i = \sum_{i \in [n]} a_i - B$. We have proved that the instance of edge interdiction problem with the number of interdicted edges is $k$ is a "yes" instance.

Next, we assume there exists an interdiction edge set $R$ with $|R| = k$ such that $\sum_{i \in R} c_i = \sum_{i \in R} w_i \leq B$ and $\sum_{i \in E \setminus R} w_i \leq \sum_{i \in [n]} a_i - B$. We will prove that the instance of $k$-Subset Sum is a "yes" instance. Due to $\sum_{i \in E \setminus R} w_i \leq \sum_{i \in [n]} a_i - B$, it implies that $\sum_{i \in R} w_i \geq B$. Since $\sum_{i \in [n]} a_i = \sum_{i \in E} w_i = \sum_{i \in E \setminus R} w_i + \sum_{i \in R} w_i$. Thus, we have $\sum_{i \in R} w_i = B$, since $c_i = w_i$ for $i \in E$. That is, $\sum_{i \in R} a_i = B$ with $|R| = k$, i.e., there exist a subset $R = S_1 \subseteq S$ with $k$ elements that sum up to $B$.

## 4 Approximation Algorithms for $B$-Matching Interdiction Problem

Since the edge interdiction problem is NP-complete even if on simple bipartite graphs with unit edge weights and unit interdiction costs [22]. To design a more efficient algorithm, we naturally consider weight partial vertex cover problem as in Zenklusen [22] and Dinitz et al. [3]. Then, we can get the results of our problem.

In the following, we consider the following problem and linear programming relaxation, denoted as $LP_{pvc}$. Assume that $S \subseteq E$ is a set of some edges. Let $c^i(S) = \sum_{e \in S} c^i_e$ be the total interdiction cost of $S$ for $i = 1, 2, \cdots, d$. In particular, we have $c^i(E) = \sum_{e \in E} c^i_e$ for $i = 1, 2, \cdots, d$.

$$\begin{cases} \min \sum\limits_{v \in V} b_v x_v \\ s.t. \sum\limits_{v \in e} x_v \geq y_e, \ \forall e \in E, \\ \sum\limits_{e \in E} c^i_e y_e \geq c^i(E) - B_i, i = 1, 2, \cdots d \\ x_v \geq 0. \ \forall v \in V, \\ 0 \leq y_e \leq 1, \ \forall e \in E. \end{cases} \tag{2}$$

**Lemma 1.** *The optimal value $OPT$ of linear programming (3) satisfies $OPT \leq OPT^*$, where $OPT^*$ is the optimal value of original b-matching interdiction problem with unit edge weight.*

*Proof.* Our proof mimics closely proof from Lemma 3 in [22]. Let $R^* \subseteq E$ be an optimal interdiction set and let $a$ be a maximum $b$-matching value in $G \setminus R^*$, i.e., follower problem optimal value in $G \setminus R^*$. Let $S$ be the optimal vertex cover set in $G \setminus R^*$. Let $d$ be a minimum weight vertex cover value in $G \setminus R^*$ with vertex weights $b_v$ for each $v \in V$, i.e., minimum weight vertex cover optimal value in $G \setminus R^*$. Due to maximum $b$-matching and minimum vertex cover problem have an integer optimal solution, we have $a = d$ by the strong duality theorem. For more details, see Chap. 18 of Schrijver's book [17]. Let $U \subseteq E$ be the set of all edges that are adjacent to at least one vertex in $S$. Consider the following solution $(x, y)$ of (3) according to the set $S$. Let $x_v = 1$, $v \in S$, otherwise, $x_v = 0$. Let $y_e = 1$, $e \in U$, otherwise, $y_e = 0$. Since $S$ is a vertex cover in $G \setminus R^*$, we have $E \setminus R^* \subseteq U$ and thus $c^i(U) \geq c^i(E) - c^i(R^*) \geq c^i(E) - B_i$ for $i = 1, 2, \cdots, d$, implying that $x$, $y$ is a feasible solution to (3). Furthermore, the objective value of (3) with the solution $x$, $y$ is equal to $d$. Since $x_v = 1$, $v \in S$. Thus, we have $OPT \leq d = a = OPT^*$.

We first discuss the definition and properties of the linear programs extreme point solution will be used in the rest of this paper. More comprehensive details can be found in the book [15]. Then, we will give the algorithm for linear programming (3).

**Definition 2.** *[15] Let $P = \{x : Ax \geq b, x \geq 0\} \subseteq \mathbb{R}^n$. Then $x \in \mathbb{R}^n$ is an* **extreme point solution** *of $P$ if there does not exist a non-zero vector $y \in \mathbb{R}^n$ such that $x + y, x - y \in P$.*

**Lemma 2.** *[15] Let $P = \{x : Ax \geq b, x \geq 0\}$ and assume that the optimum value $\min\{c^T x : x \in P\}$ is finite. Then for any feasible solution $x \in P$, there exists an extreme point solution $x' \in P$ with $c^T x' \leq c^T x$.*

**Lemma 3. (Rank Lemma)** *[15] Let $P = \{x : Ax \geq b, x \geq 0\}$ be a polytope and let $x = (x_1, \cdots, x_n)$ be an extreme point solution of $P$ such that $x_i > 0$ for each $i = 1, \cdots, n$. Then any maximal number of linearly independent tight constraints of the form $A_i x = b_i$ for some row $i$ of $A$ equals the number of variables.*

**Theorem 3.** *[16] There is an algorithm which returns an optimal extreme point solution to a linear program. Moreover, the running time of the algorithm is polynomial in the size of the linear program.*

In the following, we will give an algorithm for linear programming (3). We first guess the vertex with the highest cost in the optimal solution, such as $v$ in the optimal solution. Then, we delete all vertexes with costs greater than $v$ and include $v$ in the solution. Since there are at most $n = |V|$ vertexes, we can consider all these cases (vertexes) and return the algorithm's solution in each case, ultimately finding the best solution.

As we proceed with the iterative algorithm, we will work with a graph where edges have only one vertex. For example, when we have a variable with $x_v = 0$, we will remove $v$ from all edges containing $v$. Such edges may contain only one vertex, i.e., a loop.

The following lemma follows from a direct application of the Rank Lemma.

**Lemma 4.** *Let $x$ be an extreme point solution to the linear program $LP_{pvc}$ with $0 < x_v < 1$ for each vertex $v$ and $0 < y_e < 1$ for all $e \in E$. Then there exist $F \subseteq E$ and $J \subseteq \{1, 2, \cdots, d\}$ such that*

*(i)* $\sum_{v \in e} x_v = y_e$ *for each* $e \in F$ *and* $\sum_{e \in E} c_e^j y_e = c^j(E) - B_j$ *for each* $j \in J$.
*(ii)* *The constraints in* $\{\sum_{v \in e} x_v = y_e : e \in F\}$ *and* $\{\sum_{e \in E} c_e^j y_e = c^j(E) - B_j : j \in J\}$ *are linearly independent.*
*(iii)* $|F| + |J| = |V| + |E|$.

The following is a specific iterative algorithm for linear programming (3).

**Lemma 5.** *Let $G$ be a graph with $|V(G)| \geq (d + 1)$. Then at least one of the following must hold.*

*(i)* *There exists a vertex $v$ with $x_v \in \{0, 1\}$.*
*(ii)* *There exists an edge $e$ with $y_e = 0$.*
*(iii)* *There exists an edge $e$ with $y_e = 1$ and therefore $x_v \geq 1/2$ for some $v \in e$.*

**Algorithm 1.** Iterative algorithm

```
1: Initialization $W \leftarrow \emptyset$
2: while $E \neq \emptyset$ do
3:    Find an optimal extreme point solution $(x, y)$ to $LP_{pvc}$.
4:    (a) If there is an edge $e \in E$ with $y_e = 0$, then remove $e$ from $G$. If there is
      a vertex $v \in V$ with $x_v = 0$, then remove $v$ from $G$ and from all the edges
      containing it, i.e, $e \leftarrow e \setminus \{v\}$ for all $e \in E$.
5:    (b) If there is a vertex $v \in V$ with $x_v \geq 1/2$, then include $v \in W$ and remove
      $v$ and all the edges incident at $v$ from $G$. Update $c^i(E) - B_i \leftarrow (c^i(E) - B_i) -$
      $\sum_{e:v\in e} c_e^i$ for $i = 1, 2, \cdots, d$.
6:    (c) If $G$ contains at most $d$ vertexes $v_1, \cdots, v_d$, then include $v_1, \cdots, v_d \in W$ and
      remove $v_1, \cdots, v_d$ and all the edges incident at $v_1, \cdots, v_d$ from $G$.
7: end while
8: return $W$
```

*Proof.* Suppose for contradiction that none of the above conditions hold. Then we have $0 < x_v < 1$ for all vertexes and $0 < y_e < 1$ for all edges. From Lemma 3.3 there exists a subset of edges $F$ such that $|F| + |J| = |E| + |V|$. Since $|F| \leq |E|$, this implies that $|V| \leq |J| \leq d$, which is a contradiction.

**Theorem 4.** *There is a $(d+1)$-approximation algorithm for the weight partial vertex cover problem.*

*Proof.* Let $(x, y)$ be an extreme optimal solution of the initial linear programming (3). Let $b(LP^{(k)})$ be the linear programming (3) optimal value for the residual graph at the end of $k$-th iteration of algorithm. Let $b(W^{(k)})$ and $b(LP^{(k-1)}) - b(LP^{(k)})$ be the cost increase of the integral solution and decrease of the fractional solution at the end of $k$-th iteration of algorithm, respectively. Let $b(LP^{(0)})$ be the initial optimal value, i.e., $b(LP^{(0)}) = OPT$, where $OPT$ is the optimal value of linear programming (3). Let $(x^{(k)}, y^{(k)})$ be the solution at the algorithm executes line 3 in $k$-th iteration. Assume that the algorithm iterates $N$ steps and will terminate. We have $b(W) = \sum_{k=1}^{N} b(W^{(k)})$ and $OPT = \sum_{k=1}^{N}[b(LP^{(k-1)}) - b(LP^{(k)})]$. Note that $b(LP^{(N)}) = 0$, since at the end of $N$-th of the algorithm, it will terminate and $E = \emptyset$ in this case.

Next, we claim that at any iteration of the algorithm, the increase of the integral solution is at most $(d+1)$ times the decrease of the fractional solution in each iteration. It follows by an inductive argument that the final integral solution is at most $(d+1)$ times the initial fractional solution. Thus, $b(W) \leq (d+1)OPT$.

Consider an arbitrary iteration $k$. Note that the optimal objective value of the linear programming (3) when executing algorithm line 3 is equal to the optimal objective value of the linear programming (3) at the end of the previous iteration.

If the algorithm executes 4(a), then

$$b(W^{(k)}) = 0,$$

since $W^{(k)} = \emptyset$

$$b(LP^{(k-1)}) - b(LP^{(k)}) \geq 0,$$

since the problem at the end of $k$-th iteration is to find a covering in the graph $G' = G \setminus \{v\}$ and remove all edges incident at $v$ from $G$ or in the graph $G' = G \setminus \{e\}$. The residual solution $(x_{res}^{(k)}, y_{res}^{(k)})$, $(x^{(k)}, y^{(k)})$ restricted to $G'$, is a feasible solution to the linear programming relaxation of the residual problem. Then, we have $b(LP_{res}^{(k)}) \geq b(LP^{(k)})$, where $b(LP_{res}^{(k)})$ is the value of feasible solution $(x_{res}^{(k)}, y_{res}^{(k)})$ in residual problem. Thus, $b(LP^{(k-1)}) - b(LP^{(k)}) \geq b(LP^{(k-1)}) - b(LP_{res}^{(k)}) = b(LP^{(k-1)}) - b(LP_3^{(k)}) = 0$, where $b(LP_3^{(k)})$ is optimal objective value of the linear programming (3) when executing algorithm line 3 in $k$-th iteration, i.e. the optimal objective value of extreme optimal solution $(x^{(k)}, y^{(k)})$ in $k$-th iteration. Since the optimal objective value of the linear programming (3) when executing algorithm line 3 is equal to the optimal objective value of the linear programming (3) at the end of the previous iteration and $x_v^{(k)} = 0$ or $y_e^{(k)} = 0$. The claim holds.

If algorithm executes line 5(b), then

$$b(W^{(k)}) = b_v,$$

since we include $v \in W$.

$$b(LP^{(k-1)}) - b(LP^{(k)}) \geq x_v^{(k)} b_v,$$

since the problem at the end of $k$-th iteration is to find a covering in the graph $G' = G \setminus \{v\}$ and remove all edges incident at $v$ from $G$. The residual solution $(x_{res}^{(k)}, y_{res}^{(k)})$, $(x^{(k)}, y^{(k)})$ restricted to $G'$, is a feasible solution to the linear programming relaxation of the residual problem. Then, we have $b(LP_{res}^{(k)}) \geq b(LP^{(k)})$, where $b(LP_{res}^{(k)})$ is the value of feasible solution $(x_{res}^{(k)}, y_{res}^{(k)})$ in residual problem. Thus, $b(LP^{(k-1)}) - b(LP^{(k)}) \geq b(LP^{(k-1)}) - b(LP_{res}^{(k)}) = b(LP^{(k-1)}) - (b(LP_3^{(k)}) - x_v^{(k)} b_v) = x_v^{(k)} b_v$. The claim holds, since $b_v \leq 2x_v^{(k)} b_v$ and $x_v^{(k)} \geq 1/2$.

Finally, the Step 6(c) of the algorithm is executed in the last step, if possible. If algorithm executes line 6(c) and we assume that there are $d$ vertexes $v_1, \cdots, v_d$ in this case, then

$$b(W^{(N)}) = b_{v_1} + \cdots + b_{v_d},$$

since we include $v_1, \cdots, v_d \in W$.

$$b(LP^{(N-1)}) - b(LP^{(N)}) = b(LP^{(N-1)}) = x_{v_1}^{(N)} b_{v_1} + \cdots + x_{v_d}^{(N)} b_{v_d},$$

since $G$ contains $d$ vertexes $v_1, \cdots, v_d$.

Since the cost of $d$ vertexes $v_1, \cdots, v_d$ are at most the cost of guessing the highest vertex $h^*$ and the guessing highest vertex includes the optimal solution, we have $b_{v_1}, \cdots, b_{v_d} \leq b_h$. Thus, $b_{h^*} + b_{v_1} + \cdots + b_{v_d} \leq (d+1)(b_{h^*} + b_{v_1} x_{v_1}^{(N)} + \cdots + b_{v_d} x_{v_d}^{(N)})$, since $x_{v_1}^{(N)}, \cdots, x_{v_d}^{(N)} > 0$ in optimal solution.

To sum up, the claim also holds.

**Theorem 5.** *Based on the Lemma 3.1 and Theorem 3.5, the above algorithm is a $(d+1)$-approximation for b-matching interdiction problem on bipartite graph with unit weight and multi-dimensional budgets.*

*Proof.* The interdiction set $R = E \setminus W$ satisfies $c^i(R) \leq B_i$ for $i = 1, 2, \cdots, d$, since $c^i(W) \geq c^i(E) - B_i$ for $i = 1, 2, \cdots, d$. Assume that the value of output of Algorithm 1 is $OUT$. Then, due to the Lemma 3.1 and Theorem 3.5, we have $OUT \leq (d+1)OPT \leq (d+1)OPT^*$. The theorem holds.

**Theorem 6.** *Based on the Lemma 3.1 and Theorem 3.5, the above algorithm is a 2-approximation for b-matching interdiction problem on bipartite graph with unit weight and multi-dimensional budgets when d is a constant.*

*Proof.* When $d$ is a constant, we can guess $d$ highest vertices $h_1^*, \cdots, h_d^*$ with $b_{v_{h_1^*}}, \cdots, b_{v_{h_d^*}}$ and $O(n^d)$ enumerations.

The relationship between increase of the integral solution and decrease of the fractional solution for algorithm executes line 4(a) and 5(b) does not change, i.e., increase of the integral solution is at most twice the decrease of the fractional solution in each iteration for algorithm executes line 4(a) and 5(b).

Without loss of generality, we assume that $b_{v_1} \leq \cdots \leq b_{v_d}$ if algorithm executes line 6(c). Since the cost of $d$ vertexes $v_1, \cdots, v_d$ are at most the cost of the guessing highest vertex $h_1^*, \cdots, h_d^*$ with $b_{v_i} \leq b_{v_{h_i^*}}$ for $i = 1, \cdots, d$. Meanwhile, the guessing highest vertex includes the optimal solution. Thus, $b_{h_1^*} + \cdots + b_{h_d^*} + b_{v_1} + \cdots + b_{v_d} \leq 2(b_{h_1^*} + \cdots + b_{h_d^*} + b_{v_1}x_{v_1}^{(N)} + \cdots + b_{v_d}x_{v_d}^{(N)})$, since $x_{v_1}^{(N)}, \cdots, x_{v_d}^{(N)} > 0$ in optimal solution.

To sum up, the increase of the integral solution is at most twice the decrease of the fractional solution in each iteration of the algorithm.

## 5 Conclusion

In this paper, we consider $b$-matching interdiction problem on bipartite graphs with unit weight and multi-dimensional budgets. Let $d$ be the dimension of the leader's budget. We show that $b$-matching interdiction problem is W[1]-hard with respect to the budget for the number of interdicted edges when $d = 2$. Then, we use iterative rounding method and propose a $(d+1)$-approximation algorithm for this problem. Finally, we also show that our iterative rounding method is a 2-approximation algorithm when $d$ is a constant. Moreover, it is interesting to consider the general graph case to design a approximation algorithm. Finally, we want to extend this problem with linear or convex piecewise-linear costs in the future.

## References

1. Afshari Rad, M., Kakhki, H.T.: Two extended formulations for cardinality maximum flow network interdiction problem. Networks **69**(4), 367–377 (2017)
2. Colson, B., Marcotte, P., Savard, G.: An overview of bilevel optimization. Ann. Oper. Res. **153**(1), 235–256 (2007)
3. Dinitz, M., Gupta, A.: Packing interdiction and partial covering problems. In: Goemans, M., Correa, J. (eds.) IPCO 2013. LNCS, vol. 7801, pp. 157–168. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36694-9_14

4. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (2012)
5. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM **19**(2), 248–264 (1972)
6. Fellows, M.R., Koblitz, N.: Fixed-parameter complexity and cryptography. In: Cohen, G., Mora, T., Moreno, O. (eds.) AAECC 1993. LNCS, vol. 673, pp. 121–131. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56686-4_38
7. Frederickson, G.N., Solis-Oba, R.: Increasing the weight of minimum spanning trees. In: SODA 1996, pp. 539–546 (1996)
8. Fulkerson, D.R., Harding, G.C.: Maximizing minimum source-sink path subject to a budget constraint. Math. Program. **13**, 116–118 (1977)
9. Huang, B., Jebara, T.: Fast $b$-matching via sufficient selection belief propagation. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 361–369. JMLR Workshop and Conference Proceedings (2011)
10. Israeli, E., Wood, R.K.: Shortest-path network interdiction. Networks **40**, 97–111 (2002)
11. Jebara, T., Shchogolev, V.: B-matching for spectral clustering. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 679–686. Springer, Heidelberg (2006). https://doi.org/10.1007/11871842_67
12. Jebara, T., Wang, J., Chang, S. F.: Graph construction and b-matching for semi-supervised learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 441–448 (2009)
13. Jeroslow, R.G.: The polynomial hierarchy and a simple model for competitive analysis. Math. Program. **32**(2), 146–164 (1985)
14. Khan, A., et al.: Efficient approximation algorithms for weighted $b$-matching. SIAM J. Sci. Comput. **38**(5), S593–S619 (2016)
15. Lau, L. C., Ravi, R., Singh, M.: Iterative Methods in Combinatorial Optimization, vol. 46. Cambridge University Press (2011)
16. Nemhauser, G.L., Wolsey L.A.: Integer and Combinatorial Optimization. Wiley-Interscience (1999)
17. Schrijver, A.: Combinatorial Optimization - Polyhedra and Effciency. Springer, New York (2005)
18. Shaw, B., Jebara, T.: Minimum volume embedding. In: Artificial Intelligence and Statistics, pp. 460–467. PMLR (2007)
19. Smith, J.C., Song, Y.: A survey of network interdiction models and algorithms. Eur. J. Oper. Res. **283**(3), 797–811 (2020)
20. Stackelberg, H.: The Theory of the Market Economy. Oxford University Press (1952)
21. Zenklusen, R., Ries, B., Picouleau, C., De Werra, D., Costa, M.C., Bentz, C.: Blockers and transversals. Discret. Math. **309**(13), 4306–4314 (2009)
22. Zenklusen, R.: Matching interdiction. Discrete Appl. Math. **158**, 1676–1690 (2010)
23. Zenklusen, R.: An O(1)-approximation for minimum spanning tree interdiction. In: IEEE 56th Annual Symposium on Foundations of Computer Science, pp. 709–728 (2015)

# Mechanism Design for Facility Location Games Under a Prelocated Facility

Genjie Qin, Qizhi Fang, and Wenjing Liu(✉)

School of Mathematical Science, Ocean University of China, No. 238, Songling Road, Qingdao 266100, Shandong, China
{qfang,liuwj}@ouc.edu.cn

**Abstract.** We study the problem of locating a new homogeneous facility under a prelocated facility. Here, a group of agents are located on the real line, each of whom has her location as private information and her cost is the (expected) distance from her location to the nearest facility. Our goal is to design mechanisms which can approximately minimize the maximum cost and the social cost while eliciting agents' private information truthfully (i.e. strategy-proof).

Based on the real-life scenarios, we consider the problem in two settings: the general setting where each agent can be located at both sides of the prelocated facility, and the special setting where all the agents are located at the same side of the prelocated facility. In the general setting, we design the best possible deterministic strategy-proof mechanism with 2-approximation and provide a lower bound of $1.5 - \epsilon(\epsilon > 0)$ for any randomized strategy-proof mechanism under the maximum cost objective. For the social cost, we obtain an upper bound of $n$ for deterministic strategy-proof mechanisms, and lower bounds of 1.5 and 1.0425 for any deterministic strategy-proof mechanism and any randomized strategy-proof mechanism, respectively. In the special setting, we further provide a randomized strategy-proof 5/3-approximate mechanism for the maximum cost and a deterministic strategy-proof $(n-1)$-approximate mechanism for the social cost.

**Keywords:** Facility location game · Approximation · Prelocated facility · Algorithmic mechanism design

## 1 Introduction

Facility location problem is an important problem in the field of optimization. Its main goal is to select the best locations to place facilities under given constraints to optimize certain objectives. Such problems usually involve engineering, operations research or urban planning, such as medical facility location, logistics center location or communication base station location.

It is also the prototypical problem used by Procaccia and Tennenholtz [14] when they introduced their highly successful agenda on approximate mechanism design without money in 2009. In the basic setting of this problem, a social

D. Du et al. (Eds.): COCOA 2024, LNCS 15435, pp. 85–106, 2025.
https://doi.org/10.1007/978-981-96-4448-3_8

planner is tasked with locating a facility or two facilities on a real line based on the location profile reported by agents to minimize the maximum cost or the social cost, and each agent's cost is the distance between her location and the nearest facility. In order to benefit herself, the agent may misreport her location to manipulate the outcome. Consequently, the social planner needs to design a mechanism that outputs an (approximately) optimal outcome while guaranteeing all the agents reporting truthfully. Since then, this problem has been extensively studied in the literature of theoretical computer science and artificial intelligence.

Most previous work focuses on the locating problem where existing facilities are not taken into account. However, in many practical applications, existing resources (such as facilities) should not be wasted. For instance, due to population growth, the government needs to build a new primary school to ensure adequate educational coverage in a city. Naturally, the government must determine the location of the new school based on not only where the school-age children live but also the locations of the prelocated primary schools.

In addition, it is also ubiquitous in the real-world that all agents are located at the same side of the prelocated facility. For example, residents in coastal cities usually have a great demand for seafood, but fishermen tend to sell their seafood at the seaside, which makes it inconvenient for the residents who live far from the seaside to buy seafood. Therefore, the government decides to build a new seafood market based on the location information of the residents. For another example, the building manager in a skyscraper usually only sets up life service facilities (such as takeaway cabinets) on the ground floor at the beginning. As the number of residents increases, the demand for facilities increases. To facilitate the residents, the manager considers to add a new service facility on a certain floor based on the locations of the residents who need the service.

In order to capture such scenarios, we study how to locate a new homogeneous facility on the basis of a prelocated facility in the following two settings:

– General setting: Each agent can be located at both sides of the prelocated facility.
– Special setting: All the agents are located at the same side of the prelocated facility.

Our objective is to design mechanisms that can elicit agents' private information truthfully and approximately optimize certain social objectives.

## 1.1 Related Work

Our work is grounded on a string of research for approximate mechanism design without money, which was initiated by Procaccia and Tennenholtz [14]. They provided upper and lower bounds on the achievable approximation ratio of strategy-proof mechanisms for one facility and two-facility problems on the real line under two types of social objectives. For the one facility game, they gave the best possible deterministic and randomized mechanisms under each social objective. For the two-facility game under the maximum cost objective, they provided a best possible deterministic mechanism and an upper bound of 5/3 and a lower bound of 1.5 for randomized strategy-proof mechanisms. In addition, they gave an upper bound of $n-2$ and a lower bound of 1.5 for deterministic strategy-proof mechanisms under the social cost objective. Later, Lu et al. [13] improved these results by a randomized strategy-proof 4-approximate mechanism in general metric spaces and a lower bound of $\frac{n-1}{2}$ for any deterministic strategy-proof mechanism on a line. Fotakis and Tzamos [11] raised the lower bound to $n-2$, which ultimately eliminated the gap between the upper bound and the lower bound on deterministic strategy-proof mechanisms in two-facility games. Alon et al. [1] considered the problem of locating a facility on networks. Much work further considers the situation where the agents have various preferences for facilities. Cheng et al. [6] considered an obnoxious facility game on the network where all agents try to be as far away from the facility as possible. Ye et al. [20] studied the problem of locating an obnoxious facility on a real line under the objectives of maximizing the sum of squares of distances (maxSOS) and maximizing the sum of distances (maxSum). Serafino and Ventre [16,17] considered building two heterogeneous facilities, where the cost of each agent is the sum of the distances between her location and the facilities she is interested in. Yuan et al. [21] studied the two-facility games with min-variant and max-variant, where each agent's cost is the distance between her location and the closer facility or the further facility respectively. Li et al. [12] improved the upper bound for the min-variant. Anastasiadis and Deligkas [7] considered the situation where each agent might want to be close to a facility, be away from a facility, or be indifferent about its presence. Fong et al. [9] studied the facility location game with fractional preference where the preference of each agent for facilities is represented by a number between 0 and 1. So far, there are numerous variations of the facility location game, such as other types of cost functions [8,10], constraints on facility locations and facilities' capacity [5,18,19,22], etc. Interested readers may refer to a detailed survey by Chan et al. [2].

As far as we know, all the previous work except [3,4,15] does not take prelocated facilities into account. Recently, Chan and Wang [4] considered modifying the structure of regions by adding a costless shortcut edge based on a prelocated facility on the real line. Chan et al. [3] further studied the problem of adding a non-zero cost shortcut or two costless shortcuts. Qin et al. [15] extended the model in [4] by building a bridge to connect two separated regions. Compared with [3,4,15], we study how to locate a new homogeneous facility based on a prelocated facility, which can be considered as another natural perspective of utilizing the prelocated facility. It is worth mentioning that our model is equivalent to the model in [4] when one endpoint of the costless shortcut edge in [4] is fixed at the prelocated facility. Therefore, all the mechanisms in our work can be used directly in [4] and the mechanism in [4] that outputs one endpoint exactly at the prelocated facility can also apply to our model.

## 1.2 Our Results

In this paper, we study the problem of locating a new homogeneous facility based on a prelocated facility on the real line. We derive upper and lower bounds on the achievable approximation ratio for strategy-proof mechanisms in two settings. The results are summarized in Table 1.

In the general setting, we design the best possible deterministic strategy-proof mechanism with 2-approximation for the maximum cost. We also prove any randomized strategy-proof mechanism has an approximation ratio of at least $1.5 - \epsilon$ for the maximum cost. For the social cost, we only obtain a loose upper bound of $n$ for deterministic strategy-proof mechanisms. We derive lower bounds of 1.5 and 1.0425 for any deterministic and randomized strategy-proof mechanism, respectively.

In the special setting, we design a randomized strategy-proof $\frac{5}{3}$-approximate mechanism for the maximum cost. We also derive an upper bound of $(n-1)$ for deterministic strategy-proof mechanism under the social cost objective.

**Table 1.** A summary of our results.

| Social Objective | General Setting | | Special Setting | |
|---|---|---|---|---|
| | Deterministic | Randomized | Deterministic | Randomized |
| Maximum cost | UB: 2<br>LB: 2 | UB: 2<br>LB: $1.5 - \epsilon$ | UB: 2<br>LB: 2 | UB: 5/3<br>LB: $1.5 - \epsilon$ |
| Social cost | UB: $n$<br>LB: 1.5 | UB: 6 [4]<br>LB: 1.0425 | UB: $n - 1$<br>LB: 1.0425 | UB: 6 [4]<br>LB: 1.0425 |

**Remarks.** Under the social cost objective, we only obtain a non-constant upper bound for deterministic strategy-proof mechanisms in both settings. However, Mechanism 3 in [4] provides an upper bound of 6 for randomized strategy-proof mechanisms in our model since this mechanism always locates one endpoint of the costless shortcut at the prelocated facility. On the other hand, Mechanism 1 in our work can be used in [4] to improve their upper bound from 3 to 2 under the maximum cost objective. Further, the proof of Theorem 8 in this paper can also be used in [4] to improve their lower bound of any randomized strategy-proof mechanism from 1.02 to 1.0425 under the social cost objective.

## 2 Model

Suppose a facility has already been located on the real line, and its location is publicly known. Without loss of generality, assume that the facility is located at $y_0 = 0$. There is a set of agents $N = \{1, 2, ..., n\}$ who need to receive service from the facility. For each agent $i \in N$, she has a private location $x_i \in \mathbb{R}$ and let

$\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be a location profile or an instance. We want to construct a new homogeneous facility on the real line and every agent can be served by the nearest one.

A deterministic mechanism is a function $f : \mathbb{R}^n \to \mathbb{R}$ which maps a location profile to a facility location. If $f(\mathbf{x}) = y$, the cost of agent $i \in N$ is $cost(x_i, f(\mathbf{x})) = \min\{|x_i|, |x_i - y|\}$. A randomized mechanism is a function $f$ which maps a location profile to a probability distribution over facility locations. Formally, $f$ can be defined as $f : \mathbb{R}^n \to \Delta(\mathbb{R})$, where $\Delta(\mathbb{R})$ represents the set of probability distributions over $\mathbb{R}$ and the cost of agent $i \in N$ is $cost(x_i, f(\mathbf{x})) = \mathbb{E}_{Y \sim f(\mathbf{x})} \min\{|x_i|, |x_i - Y|\}$.

A mechanism is strategy-proof if no agent can benefit from misreporting her location regardless of the locations reported by the others. Formally, $\forall \mathbf{x} \in \mathbb{R}^n, \forall i \in N, \forall x_i' \in \mathbb{R}, cost(x_i, f(\mathbf{x})) \le cost\left(x_i, f(x_i', \mathbf{x}_{-i})\right)$. Here, $\mathbf{x}_{-i}$ represents the location vector of the set of agents $N \backslash \{i\}$.

The maximum cost and the social cost of a mechanism $f$ with respect to a location profile $\mathbf{x}$ are defined as the maximum cost among all $n$ agents and the total cost of all $n$ agents, respectively:

$$MC(\mathbf{x}, f) = \max \_i \in N cost(x_i, f(\mathbf{x})), SC(\mathbf{x}, f) = \sum_{i \in N} cost(x_i, f(\mathbf{x}))$$

Let the optimal solution for an instance $\mathbf{x}$ under the maximum cost objective or the social cost objective be denoted as $OPT_{MC}(\mathbf{x})$ and $OPT_{SC}(\mathbf{x})$, respectively. The subscript is omitted without confusion. A strategy-proof mechanism $f$ has an approximation ratio of $\gamma(\ge 1)$ under the maximum cost objective, if

$$\gamma = \sup_{\mathbf{x} \in \mathbb{R}^n} \frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT_{MC})}$$

The approximation ratio is defined similarly under the social cost objective. In this paper, we will focus on anonymous[1] strategy-proof mechanisms with good approximation ratios under the social objectives of either minimizing the maximum cost or minimizing the social cost. Henceforth, for simplicity, the instance $\mathbf{x}$ is assumed to satisfy $x_1 \le x_2 \le ... \le x_n$ without further comment.

**Notations.** For $\forall \mathbf{x} \in \mathbb{R}^n$, $x_1$ and $x_n$ are the two endpoints. Define $L(\mathbf{x})$ and $S(\mathbf{x})$ as the locations of the endpoints that are farther and closer to $y_0$, respectively. Define

$$l(\mathbf{x}) = \begin{cases} \min\left\{x_i | x_i > \frac{L(\mathbf{x})}{3}\right\}, \text{if } L(\mathbf{x}) \ge 0 \\ \max\left\{x_i | x_i < \frac{L(\mathbf{x})}{3}\right\}, \text{if } L(\mathbf{x}) < 0 \end{cases}$$

$$b(\mathbf{x}) = \begin{cases} \max\left\{x_i | x_i \le \frac{L(\mathbf{x})}{3}\right\}, \text{if } L(\mathbf{x}) \ge 0 \\ \min\left\{x_i | x_i \ge \frac{L(\mathbf{x})}{3}\right\}, \text{if } L(\mathbf{x}) < 0 \end{cases}$$

[1] A mechanism is anonymous if its outcome depends only on the agents' locations, not on their identities.

## 3 General Setting

This section discusses how to locate a new (homogeneous) facility on the assumption that each agent can be at both sides of the prelocated facility. We will study deterministic and randomized strategy-proof mechanisms under the objectives of minimizing the maximum cost and the social cost respectively.

### 3.1 Maximum Cost

For the maximum cost, we first present the optimal solution and the optimal value. Then we design a deterministic strategy-proof 2-approximate mechanism which is also the best possible deterministic mechanism. We also prove a lower bound of $1.5 - \epsilon$ for any randomized strategy-proof mechanism.

**Theorem 1.** *For $\forall \mathbf{x} \in \mathbb{R}^n$, the optimal solution for minimizing the maximum cost is given by*

$$OPT(\mathbf{x}) = \frac{l(\mathbf{x}) + L(\mathbf{x})}{2}.$$

*The optimal value is*

$$MC(\mathbf{x}, OPT) = \begin{cases} \max\left\{|S(\mathbf{x})|, |b(\mathbf{x})|, \frac{|L(\mathbf{x})-l(\mathbf{x})|}{2}\right\}, & if S(\mathbf{x}) \cdot L(\mathbf{x}) < 0 \\ \max\left\{|b(\mathbf{x})|, \frac{|L(\mathbf{x})-l(\mathbf{x})|}{2}\right\}, & if S(\mathbf{x}) \cdot L(\mathbf{x}) \geq 0 \end{cases}$$

Note that the optimal mechanism is not strategy-proof. For example, consider an instance $\mathbf{x} = (x_1, x_2) = (-3, 4)$. Obviously, $OPT(\mathbf{x}) = 4, cost(x_1, OPT(\mathbf{x})) = 3$. However, if agent 1 misreports her location as $-5$, denoting $\mathbf{x}' = (-5, 4)$, we have $cost(x_1, OPT(\mathbf{x}')) = cost(x_1, -5) = 2$. Thus, agent 1 can decrease her cost by misreporting.

**Theorem 2.** *Any deterministic strategy-proof mechanism has an approximation ratio of at least 2 for the maximum cost.*

**Mechanism 1.** *Given $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n$, if $|x_n| \geq |x_1|$, the facility is located at $y$ as follows:*

$$y = \begin{cases} x_n & if\, 0 \leq x_1 \leq x_n \\ \max\{2|x_1|, x_n\} & if\, x_1 < 0 \leq x_n. \end{cases}$$

*If $|x_n| < |x_1|$, $y$ is defined symmetrically.*

**Theorem 3.** *Mechanism 1 is strategy-proof.*

*Proof.* Denote Mechanism 1 by $f$. Given any instance $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$, we need to prove that every agent $i \in N$ cannot benefit by misreporting her location $x_i$ as $x_i' \in \mathbb{R}$. Denote $\mathbf{x}' = (x_i', \mathbf{x}_{-i})$. The proof falls into the following cases:

**Case 1.** $0 \leq x_1 \leq x_n$.

Obviously, $f(\mathbf{x}) = x_n$ and any agent $i$ with $x_i = x_n$ has no incentive to lie. Now consider agent $i$ with $x_i < x_n$. If $x_i' > x_i$, then $f(\mathbf{x}') \geq f(\mathbf{x})$, which implies that the facility is moving further away from agent $i$. If $x_i' < x_i$, then $f(\mathbf{x}') = f(\mathbf{x})$ when $x_i' \geq -\frac{x_n}{2}$, $f(\mathbf{x}') = 2|x_i'| \geq f(\mathbf{x})$ when $-x_n \leq x_i' < -\frac{x_n}{2}$ and $f(\mathbf{x}') \leq x_i' < -x_n$ when $x_i' < -x_n$, none of which can make agent $i$'s cost decrease.

**Case 2.** $x_1 < 0 \leq x_n$ and $2|x_1| \leq x_n$.

In this case, $f(\mathbf{x}) = x_n$. For agent $i$ with $x_i \geq 0$, the proof is similar to Case 1. Now consider agent $i$ with $x_i < 0$. Here, $cost(x_i, f(\mathbf{x})) = |x_i| \leq x_1 \leq \frac{x_n}{2}$. Note that $f(\mathbf{x}') \geq f(\mathbf{x})$ when $x_i' > -x_n$ and $f(\mathbf{x}') \leq x_i'$ when $x_i' < -x_n$. Both of them keep agent $i$'s cost unchanged.

**Case 3.** $x_1 < 0 \leq x_n$ and $2|x_1| > x_n$.

In this case, $f(\mathbf{x}) = 2|x_1|$. For agent $i$ with $x_i \geq 0$, her cost will never decrease by misreporting. For agent $i$ with $x_i < 0$, $cost(x_i, f(\mathbf{x})) = |x_i| \leq |x_1| \leq x_n$. If $x_i' \geq -x_n$, then $f(\mathbf{x}') \geq x_n \geq 0$. Otherwise, $f(\mathbf{x}') = -\max\{2x_n, |x_i'|\} \leq -2x_n$, which implies $|x_i - f(\mathbf{x}')| \geq x_n \geq |x_i|$. In both cases, the cost of agent $i$ have not changed.

**Theorem 4.** *Mechanism 1 is 2-approximate under the maximum cost objective.*

*Proof.* Denote Mechanism 1 by $f$. Without loss of generality, consider any instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ with $|x_n| \geq |x_1|$.

If $0 \leq x_1 \leq x_n$, then $f(\mathbf{x}) = x_n$,

$$\frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} \leq \frac{\max\{b(\mathbf{x}), x_n - l(\mathbf{x})\}}{\max\left\{b(\mathbf{x}), \frac{x_n - l(\mathbf{x})}{2}\right\}} \leq 2$$

If $x_1 < 0 \leq x_n, 2|x_1| \leq x_n$, then $f(\mathbf{x}) = x_n$,

$$\frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} \leq \frac{\max\{|x_1|, b(\mathbf{x}), x_n - l(\mathbf{x})\}}{\max\left\{|x_1|, b(\mathbf{x}), \frac{x_n - l(\mathbf{x})}{2}\right\}} \leq 2$$

If $x_1 < 0 \leq x_n, 2|x_1| > x_n$, then $f(\mathbf{x}) = 2x_1$,

$$\frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} \leq \frac{\max\left\{|x_1|, \frac{x_n - l(\mathbf{x})}{2}\right\}}{\max\{|x_1|, |x_n|\}} = \frac{|x_1|}{\max\{|x_1|, |x_n|\}} = 1$$

The proof is completed.

**Theorem 5.** *Any randomized strategy-proof mechanism has an approximation ratio of at least $1.5 - \epsilon$ for the maximum cost, $\epsilon > 0$.*

**Due to length limitation, part of the proofs are included in Appendix.**

## 3.2 Social Cost

This subsection is dedicated to the discussion of the social cost objective. We prove that Mechanism 1 is $n$-approximate for the social cost. We obtain lower bounds of 1.5 and 1.0425 for any deterministic and randomized strategy-proof mechanism, respectively.

**Lemma 1.** *Denote Mechanism 1 by $f$. For any instance $\mathbf{x} = (x_1, ..., x_n)$ with $0 \leq x_1 \leq x_n$,*

$$SC(\mathbf{x}, f) \leq (n-1) \cdot SC(\mathbf{x}, OPT)$$

**Theorem 6.** *Mechanism 1 is $n$-approximate under the social cost objective.*

**Theorem 7.** *Any deterministic strategy-proof mechanism has an approximation ratio of at least* 1.5 *for the social cost.*

Before our proof of the lower bound for any randomized strategy-proof mechanism, recall that strategy-proofness is equivalent to partial group strategy-proofness for facility location games where each agent has her location as private information [13]. In other words, for any group of agents located at the same location, no member can benefit if they misreport simultaneously.

**Theorem 8.** *Any randomized strategy-proof mechanism has an approximation ratio of at least* 1.0425 *for the social cost.*

*Proof.* Assume there exists a randomized strategy-proof mechanism $f$ with an approximation ratio less than 1.0425. Let $\mathbf{x}$ be an instance with 7 agents, where 4 agents are located at $x_1 = 0.7$ and 3 agents are located at $x_2 = 2$. It is clear that $OPT(\mathbf{x}) = 2$ and $SC(\mathbf{x}, OPT) = 2.8$. Assume the mechanism $f$ outputs $Y \geq 1.4$ with probability $p$. Since $E_{Y \sim f(\mathbf{x})}\left[SC(\mathbf{x}, Y) | Y < 1.4\right] \geq SC(\mathbf{x}, 0.7) = 3.9$,

$$SC(\mathbf{x}, f) \geq p \cdot 2.8 + (1-p) \cdot 3.9 = 3.9 - 1.1 \cdot p$$
$$\frac{SC(\mathbf{x}, f)}{SC(\mathbf{x}, OPT)} < 1.0425 \Rightarrow 3.9 - 1.1 \cdot p < 1.0425 \times 2.8 \Rightarrow p > \frac{981}{1100}$$

Then the cost of agent 1 is

$$cost(x_1, f(\mathbf{x})) > p \cdot 0.7 + (1-p) \cdot 0 > \frac{6867}{11000} > 0.62427$$

Now consider another instance $\mathbf{x}'$ with 7 agents, where 4 agents are located at $x_1' = 1$ and 3 agents are located at $x_2 = 2$. It is clear that $OPT(\mathbf{x}') = 1$ and $SC(\mathbf{x}', OPT) = 3$. Suppose the mechanism $f$ outputs $Y' \in [0.17, 1.23]$ with probability $q$ for instance $\mathbf{x}'$. Since $E_{Y' \sim f(\mathbf{x}')}\left[SC(\mathbf{x}', Y') | Y' \notin [0.17, 1.23]\right] \geq SC(\mathbf{x}', 1.23) = 4 \times 0.23 + 3 \times 0.77 = 3.23$,

$$SC(\mathbf{x}', f) \geq q \cdot 3 + (1-q) \cdot 3.23 = 3.23 - 0.23 \cdot \text{ q}$$

Given that the approximation ratio of any randomized strategy-proof mechanism is less than 1.0425,

$$\frac{SC(\mathbf{x}', f)}{SC(\mathbf{x}', OPT)} < 1.0425 \Rightarrow 3.23 - 0.23 \cdot q < 1.0425 \times 3 \Rightarrow q > \frac{41}{92}$$

$$cost(x_1, f(\mathbf{x}')) < q \cdot 0.53 + (1-q) \cdot 0.7 < \frac{5743}{9200} < 0.62427$$

Thus,

$$cost(x_1, f(\mathbf{x}')) < cost(x_1, f(\mathbf{x})),$$

which contradicts the strategy-proofness of the mechanism.

## 3.3 Discussion

For the social cost, there is a huge gap between the upper bound of $n$ and the lower bound of 1.5 for deterministic strategy-proof mechanisms. We conjecture the deterministic lower bound is $\Omega(n)$ just as in two-facility location games [11], but failed to verify it.

It is noteworthy that Mechanism 3 in [4] outputs one endpoint of the shortcut edge at point 0 and the other at $x_k$ with probability $\frac{|x_k|}{\sum_{i \in N} |x_i|}, k \in N$, which is proved to be strategy-proof and 6-approximate under the social cost objective. This can be described in our model as follows.

**Mechanism 2.** [4] For $\forall \mathbf{x}$, for any agent $k$, the mechanism outputs $x_k$ with probability $\frac{|x_k|}{\sum_{i \in N} |x_i|}$.

**Theorem 9.** [4] Mechanism 2 is a randomized group strategy-proof 6-approximate mechanism for the social cost.

In addition, the instances and methods in the proof of Theorem 8 can be used in [4] to improve the lower bound of any randomized strategy-proof mechanism under the social cost objective from 1.02 to 1.0425.

## 4 Special Setting

This section considers the scenario where all the agents are located at the same side of the prelocated facility. Without loss of generality, assume all of them are on the right side of $y_0 = 0$, i.e., any instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ satisfies $0 \le x_1 \le x_2 \le ... \le x_n$. Then $L(\mathbf{x}) = x_n, S(\mathbf{x}) = x_1$, the optimal solution for minimizing the maximum cost is $OPT(\mathbf{x}) = \frac{l(\mathbf{x})+L(\mathbf{x})}{2}$, and the optimal value is

$$MC(\mathbf{x}, OPT) = \max\left\{b(\mathbf{x}), \frac{|L(\mathbf{x}) - l(\mathbf{x})|}{2}\right\}.$$

In this section, the impossibility results in Sect. 3 still hold, except for that of the deterministic mechanism for the social cost. Mechanism 1 which outputs

$L(\mathbf{x}) = x_n$ for any instance $\mathbf{x}$ in this setting, is still 2-approximate and the best deterministic strategy-proof mechanism for the maximum cost. Furthermore, we provide a randomized strategy-proof 5/3-approximate mechanism for the maximum cost.

**Mechanism 3.** *For an instance $\mathbf{x} = (x_1, \ldots, x_n)$, the facility is located at $y$, where $y$ is defined as a random point according to the following cases:*

***Case 1.*** $b(\mathbf{x}) \geq L(\mathbf{x}) - l(\mathbf{x})$. *Let*

$$y = \begin{cases} L(\mathbf{x}) - b(\mathbf{x}) & \text{, with probability (w.p.) } \frac{1}{6} \\ \frac{2L(\mathbf{x}) - b(\mathbf{x})}{2} & \text{, w.p. } \frac{1}{3} \\ L(\mathbf{x}) & \text{, w.p. } \frac{1}{2} \end{cases}$$

***Case 2.*** $b(\mathbf{x}) < L(\mathbf{x}) - l(\mathbf{x})$. *Let*

$$y = \begin{cases} \max\left\{l(\mathbf{x}), \frac{2L(\mathbf{x})}{3}\right\} & \text{, with probability (w.p.) } \frac{1}{6} \\ \frac{\max\left\{l(\mathbf{x}), \frac{2L(\mathbf{x})}{3}\right\} + L(\mathbf{x})}{2} & \text{, w.p. } \frac{1}{3} \\ L(\mathbf{x}) & \text{, w.p. } \frac{1}{2} \end{cases}$$

**Theorem 10.** *Mechanism 3 is a randomized strategy-proof $\frac{5}{3}$-approximate mechanism under the maximum cost objective.*

For the social cost objective, according to Lemma 1, we have the following theorem.

**Theorem 11.** *Mechanism 1 is a deterministic strategy-proof $(n-1)$-approximate mechanism under the social cost objective.*

## 5 Conclusions and Open Problems

In this paper, we studied locating a new homogeneous facility on the real line based on the prelocated facility, where each agent has her location as private information and can be served by the nearest facility. We derived upper and lower bounds on the approximation ratio for deterministic and randomized strategy-proof mechanisms in two settings: the general setting where each agent can be located at both sides of the prelocated facility, and the special setting where all the agents are located at the same side of the prelocated facility.

For the general setting, we provided the best possible deterministic strategy-proof mechanism with 2-approximation and a lower bound of $1.5 - \epsilon$ for any randomized strategy-proof mechanism under the maximum cost objective. Under the social cost objective, we only obtained a non-constant upper bound of $n$ for deterministic strategy-proof mechanisms. In addition, we gave lower bounds of 1.5 and 1.0425 for any deterministic and randomized strategy-proof mechanism, respectively. For the special setting, we further derived an upper bound of 5/3 for randomized strategy-proof mechanisms under the maximum cost objective, and $n-1$ for deterministic strategy-proof mechanisms under the social cost objective.

For future work, a natural direction is to narrow down the gap between the upper bound and the lower bound for strategy-proof mechanisms in our model. Our model can also be extended to other settings with various facility preferences, such as obnoxious facilities, or where agents have different preferences between the prelocated facility and the newly established one. It is also worthwhile to investigate locating new facilities under a certain constraint on facilities.

**Acknowledgments.** This research was supported in part by the National Natural Science Foundation of China (12201590, 12171444).

## Appendix

### 1 Proof of Theorem 1

*Proof.* Consider the case of $S(\mathbf{x}) \cdot L(\mathbf{x}) < 0$. Without loss of generality, let $S(\mathbf{x}) \leq 0 \leq L(\mathbf{x})$. Obviously, $MC(\mathbf{x}, OPT) \geq |S(\mathbf{x})|$. If $OPT(\mathbf{x}) \geq \frac{2L(\mathbf{x})}{3}$, then the cost of agent on $b(\mathbf{x})$ is $b(\mathbf{x})$. Furthermore, the maximum cost of the agents on $l(\mathbf{x})$ and $L(\mathbf{x})$ is at least $\frac{L(\mathbf{x})-l(\mathbf{x})}{2}$. Combining the above, we have $MC(\mathbf{x}, OPT) \geq \max\left\{|S(\mathbf{x})|, |b(\mathbf{x})|, \frac{|L(\mathbf{x})-l(\mathbf{x})|}{2}\right\}$. It is easy to see that $\frac{l(\mathbf{x})+L(\mathbf{x})}{2}$ achieves a maximum cost of at most $\max\left\{|S(\mathbf{x})|, |b(\mathbf{x}), \frac{|L(\mathbf{x})-l(\mathbf{x}))|}{2}|\right\}$, which indicates the optimality. The proof for the case of $S(\mathbf{x}) \cdot L(\mathbf{x}) \geq 0$ is similar.

### 2 Proof of Theorem 2

*Proof.* Assume there exists a deterministic strategy-proof mechanism $f$ with approximation ratio less than 2. Consider an instance $\mathbf{x} = (x_1, x_2) = (1, 2)$. It holds that $MC(\mathbf{x}, f) < 2 \cdot MC(\mathbf{x}, OPT) = 1$. Therefore, $f(\mathbf{x}) \in (1, 2)$. Without loss of generality, let $f(\mathbf{x}) = 1 + \epsilon$ where $\epsilon \in (0, \frac{1}{2}]$. Consider the instance $\mathbf{x}' = (x_1, x_2') = (1, 1+\epsilon)$, then $f(\mathbf{x}') \in (1, 1+\epsilon)$. Agent 2 can benefit by reporting $x_2'$ to $x_2$, which contradicts the strategy-proofness of $f$.

### 3 Proof of Theorem 5

*Proof.* Let $f$ be any randomized strategy-proof mechanism. Let $M > 0$ be sufficiently large. Consider an instance $\mathbf{x} = (x_1, x_2) = (M + 1, M + 2)$. Obviously, $\sum_{i\in\{1,2\}} cost(x_i, f(\mathbf{x})) = E_{Y\sim f(\mathbf{x})}[cost(x_1, Y) + cost(x_2, Y)] \geq 1$. Without loss of generality, assume that $cost(x_1, f(\mathbf{x})) \geq \frac{1}{2}$.

Now consider another instance $\mathbf{x}' = (x_1', x_2') = (M, M + 2)$. Note that $OPT(\mathbf{x}') = M + 1$ and $MC(\mathbf{x}', OPT) = 1$. By strategy-proofness, we have $cost(M + 1, f(\mathbf{x}')) = cost(x_1, f(\mathbf{x}')) \geq cost(x_1, f(\mathbf{x})) \geq \frac{1}{2}$. Otherwise, the agent located at $x_1$ in $\mathbf{x}$ can benefit by misreporting her location as $x_1' = M$. The maximum cost of $f$ w.r.t. $\mathbf{x}'$ is

$$\begin{aligned}MC(\mathbf{x}', f) &= E_{Y' \sim f(\mathbf{x}')}\left[\max_{i\in\{1,2\}} cost(x'_i, Y')\right]\\ &= Pr\{Y' \le 2M\} \cdot E_{Y' \sim f(\mathbf{x}')}\left[\max_{i\in\{1,2\}} cost(x'_i, Y')|Y' \le 2M\right]\\ &+ Pr\{Y' > 2M\} \cdot E_{Y' \sim f(\mathbf{x}')}\left[\max_{i\in\{1,2\}} cost(x'_i, Y')|Y' > 2M\right]\end{aligned}$$

Note that $\max_{i\in\{1,2\}} cost(x'_i, Y') = 1 + cost(M + 1, Y')$ when $Y' \le 2M$ and $\max_{i\in\{1,2\}} cost(x'_i, Y') > M \ge cost(M + 1, Y') - 1$ when $Y' > M$. We analyze $MC(\mathbf{x}', f)$ according to the following cases.

**Case 1.** $Pr\{Y' > 2M\} \ge \frac{3}{2M}$.

$$MC(\mathbf{x}', f) \ge Pr\{Y' > 2M\} \cdot E_{Y' \sim f(\mathbf{x}')}\left[\max_{i\in\{1,2\}} cost(x'_i, Y')|Y' > 2M\right] \ge \frac{3}{2M} \cdot M = \frac{3}{2}$$

**Case 2.** $Pr\{Y' < 2M\} \ge \frac{3}{2M}$.

$$\begin{aligned}MC(\mathbf{x}', f) &\ge Pr\{Y' \le 2M\} \cdot E_{Y' \sim f(\mathbf{x}')}[1 + cost(M + 1, Y')|Y' \le 2M]\\ &+ Pr\{Y' > 2M\} \cdot E_{Y' \sim f(\mathbf{x}')}[1 + cost(M + 1, Y') - 2|Y' > 2M]\\ &= 1 + cost(M + 1, Y') - 2Pr\{Y' > 2M\}\\ &\ge \frac{3}{2} - \frac{3}{M}\end{aligned}$$

Therefore,

$$\frac{MC(\mathbf{x}', f)}{MC(\mathbf{x}', OPT)} \ge \frac{3}{2} - \frac{3}{M}$$

## 4 Proof of Lemma 1

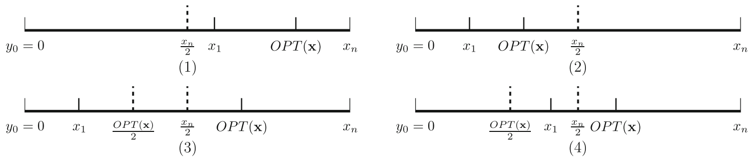


**Fig. 1.** An instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ with $0 \le x_1 \le x_n$

*Proof.* Note that $x_1 \le OPT(\mathbf{x}) \le x_n$ and $f(\mathbf{x}) = x_n$. The proof is performed according to the following cases:

**Case 1.** $x_1 \ge \frac{x_n}{2}$, as shown in Fig. 1(1).

It is clear that $OPT(\mathbf{x}) = x_{\lfloor \frac{n+1}{2} \rfloor}$ and $cost(x_i, f(\mathbf{x})) = x_n - x_i, \forall i \in N$. We have

$$SC(\mathbf{x}, OPT) = \sum_{i \in N} \left| x_i - x_{\lfloor \frac{n+1}{2} \rfloor} \right| \geq |x_1 - x_{\lfloor \frac{n+1}{2} \rfloor}| + |x_n - x_{\lfloor \frac{n+1}{2} \rfloor}| = x_n - x_1$$

$$SC(\mathbf{x}, f) = \sum_{i \in N} cost(x_i, f(\mathbf{x})) \leq \sum_{i=1}^{n-1} (x_n - x_1) \leq (n-1) \cdot SC(\mathbf{x}, OPT)$$

**Case 2.** $x_1 < \frac{x_n}{2}$.

**Case 2.1.** $OPT(\mathbf{x}) \leq \frac{x_n}{2}$, as shown in Fig. 1(2).

$$SC(\mathbf{x}, OPT) \geq cost(x_n, OPT(\mathbf{x})) = x_n - OPT(\mathbf{x}) \geq \frac{x_n}{2}$$
$$SC(\mathbf{x}, f) = \sum_{i \in N} cost(x_i, f(\mathbf{x})) = \sum_{i: x_i \leq \frac{x_n}{2}} x_i + \sum_{i: x_i > \frac{x_n}{2}} (x_n - x_i) \leq (n-1) \cdot SC(\mathbf{x}, OPT)$$

**Case 2.2.** $OPT(\mathbf{x}) > \frac{x_n}{2}$ and $x_1 < \frac{OPT(\mathbf{x})}{2}$, as shown in Fig. 1(3).

Partition $N$ as follows:

$$N_1 = \left\{ i \in N \middle| x_1 \leq x_i \leq \frac{1}{2} OPT(\mathbf{x}) \right\}, N_2 = \left\{ i \in N \middle| \frac{1}{2} OPT(\mathbf{x}) < x_i \leq \frac{x_n}{2} \right\}$$
$$N_3 = \left\{ i \in N \middle| \frac{x_n}{2} < x_i \leq OPT(\mathbf{x}) \right\}, N_4 = \left\{ i \in N \middle| OPT(\mathbf{x}) < x_i \leq x_n \right\}$$

Thus, we have

$$\begin{aligned}
SC(\mathbf{x}, OPT) \geq & \sum_{i \in N_1 \cup N_2 \cup N_3} cost(x_i, f(\mathbf{x})) + (x_n - OPT(\mathbf{x})) \\
SC(\mathbf{x}, f) = & \sum_{i \in N_1} cost(x_i, OPT(\mathbf{x})) + \sum_{i \in N_2} cost(x_i, f(\mathbf{x})) \\
& + \sum_{i \in N_3} [cost(x_i, OPT(\mathbf{x})) + x_n - OPT(\mathbf{x})] + \sum_{i \in N_4} cost(x_i, f(\mathbf{x})) \\
\leq & \sum_{i \in N_1} cost(x_i, OPT(\mathbf{x})) + \sum_{i \in N_2} \left[ \frac{x_n}{2} + cost(x_i, OPT(\mathbf{x})) - OPT(\mathbf{x}) + \frac{x_n}{2} \right] \\
& + \sum_{i \in N_3} [cost(x_i, OPT(\mathbf{x})) + x_n - OPT(\mathbf{x})] + \sum_{i \in N_4 - \{x_n\}} [x_n - OPT(\mathbf{x})] \\
\leq & \sum_{i \in N_1 \cup N_2 \cup N_3} cost(x_i, OPT(\mathbf{x})) + |N_2 \cup N_3 \cup N_4 - \{x_n\}| \cdot (x_n - OPT(\mathbf{x})) \\
\leq & SC(\mathbf{x}, OPT) + (n-2) \cdot (x_n - OPT(\mathbf{x})) \\
\frac{SC(\mathbf{x}, f)}{SC(\mathbf{x}, OPT)} \leq & \frac{SC(\mathbf{x}, OPT) + (n-2) \cdot (x_n - OPT(\mathbf{x}))}{SC(\mathbf{x}, OPT)} \leq 1 + \frac{(n-2) \cdot (x_n - OPT(\mathbf{x}))}{x_n - OPT(\mathbf{x})} = n - 1
\end{aligned}$$

**Case 2.3.** $OPT(\mathbf{x}) > \frac{x_n}{2}$ and $x_1 > \frac{OPT(\mathbf{x})}{2}$, as shown in Fig. 1(4).

$$SC(\mathbf{x}, OPT) \geq cost(x_1, OPT(\mathbf{x})) + cost(x_n, OPT(\mathbf{x})) = x_n - x_1 > \frac{x_n}{2}$$

It is trivial that

$$SC(\mathbf{x}, f) \leq (n-1) \cdot SC(\mathbf{x}, OPT)$$

## 5 Proof of Theorem 6

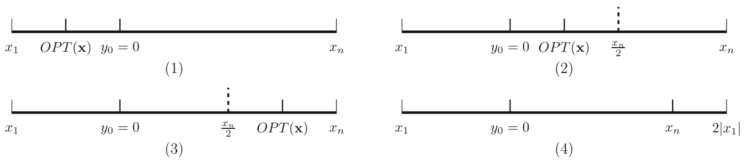


**Fig. 2.** An instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ with $x_1 < 0 \le x_n$

*Proof.* Without loss of generality, given any instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ with $|x_n| \ge |x_1|$. Trivially, $x_1 \le OPT(\mathbf{x}) \le x_n$. Denote Mechanism 1 by $f$, and our analysis is as follows.

**Case 1.** $0 \le x_1 \le x_n$. By Lemma 1, we have

$$SC(\mathbf{x}, f) \le (n-1) \cdot SC(\mathbf{x}, OPT)$$

**Case 2.** $x_1 < 0 \le x_n$ and $|2x_1| \le x_n$. We have $f(\mathbf{x}) = x_n$ and

$$SC(\mathbf{x}, f) = \sum_{i \in N} cost(x_i, f(\mathbf{x})) = \sum_{i: x_i \le \frac{x_n}{2}} |x_i| + \sum_{i: x_i > \frac{x_n}{2}} (x_n - x_i) \qquad (*)$$

**Case 2.1.** $OPT(\mathbf{x}) \le 0$, as shown in Fig. 2(1).
Note that $SC(\mathbf{x}, OPT) \ge cost(x_n, OPT(\mathbf{x})) = x_n$. Thus, by $(*)$, we have

$$SC(\mathbf{x}, f) \le (n-1) \cdot \frac{x_n}{2} \le \frac{n-1}{2} \cdot SC(\mathbf{x}, OPT)$$

**Case 2.2.** $0 < OPT(\mathbf{x}) \le \frac{x_n}{2}$, as shown in Fig. 2(2).
Since $SC(\mathbf{x}, OPT) \ge cost(x_n, OPT(\mathbf{x})) \ge \frac{x_n}{2}$, then by $(*)$,

$$SC(\mathbf{x}, f) \le (n-1) \cdot SC(\mathbf{x}, OPT)$$

**Case 2.3.** $OPT(\mathbf{x}) > \frac{x_n}{2}$, as shown in Fig. 2(3).
By a similar proof as Case 2.2 in Lemma 1, we can obtain

$$SC(\mathbf{x}, f) \le (n-1) \cdot SC(\mathbf{x}, OPT)$$

**Case 3.** $x_1 < 0 \le x_n, |2x_1| > x_n$, as shown in Fig. 2(4).
In this case, $f(\mathbf{x}) = 2|x_1|$. Note that $SC(\mathbf{x}, OPT) \ge |x_1|$. We have

$$SC(\mathbf{x}, f) = \sum_{i \in N} cost(x_i, f(\mathbf{x})) = \sum_{i: x_i \le |x_1|} |x_i| + \sum_{i: x_i > x_1} (2x_1 - x_i) \le n \cdot |x_1| \le n \cdot SC(\mathbf{x}, OPT)$$

**Tight Example.** Let $\mathbf{x}$ be an instance with $n$ agents, where $n-1$ agents are located at 1 and one agent is located at $-1$. It is clear that $OPT(\mathbf{x}) = 1, SC(\mathbf{x}, OPT) = 1$. Since $f(\mathbf{x}) = 2, SC(\mathbf{x}, f) = n - 1 + 1 = n$, we have $SC(\mathbf{x}, f) = n \cdot SC(\mathbf{x}, OPT)$.

## 6 Proof of Theorem 7

*Proof.* Assume there exists a deterministic strategy-proof mechanism $f$ with approximation ratio less than 1.5. Consider an instance $\mathbf{x} = (x_1, x_2) = (-1, 1)$. Without loss of generality, let $f(\mathbf{x}) = y \leq 0$, we have $cost(x_2, y) = 1$. Consider the instance $\mathbf{x}' = (x_1, x_2') = (-1, 1.5)$ and $f(\mathbf{x}') = y'$, we have $SC(\mathbf{x}', y') < 1.5 \cdot SC(\mathbf{x}', OPT) = 1.5$. It means $y' > 0$, we have $cost(x_2', y') = SC(\mathbf{x}', y') - cost(x_1, y') < \frac{1}{2}$. It can lead to the $cost(x_2, y') < 1$. Agent 2 can benefit by reporting $x_2$ to $x_2'$, which contradicts with the strategy-proofness of $f$.

## 7 Proof of Theorem 10

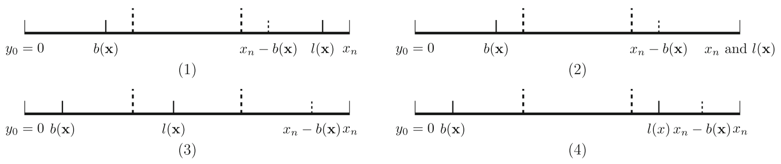


**Fig. 3.** An instance $\mathbf{x} = (x_1, x_2, ..., x_n)$ with $0 \leq x_1 \leq x_n$

*Proof.* Denote Mechanism 3 by $f$.

**Strategy-Proofness.** We need to prove that for any instance $\mathbf{x}$, every agent $i \in N$ cannot benefit by misreporting her location $x_i$ as $x_i' \in \mathbb{R}$. Denote $\mathbf{x}' = (x_i', \mathbf{x}_{-i})$ and the possible locations of $f(\mathbf{x})$ as $A$, $\frac{A+B}{2}$, $B$ from left to right. Our analysis falls into the following cases:

**Case 1.** $b(\mathbf{x}) \geq L(\mathbf{x}) - l(\mathbf{x})$ and $l(\mathbf{x}) \neq L(\mathbf{x})$, as shown in Fig. 3(1). Suppose agent $i$ reports her location $x_i$ to $x_i'$.

**Case 1.1.** $x_i \leq b(\mathbf{x})$. Agent $i$ always choose the prelocated facility, then we have $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$.

**Case 1.2.** $x_i \in [l(\mathbf{x}), L(\mathbf{x}))$.

**Case 1.2.1.** $x_i' \leq L(\mathbf{x})$. The possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A'$, $\frac{A'+B'}{2}$, $B'$ from left to right. We have $A' \in \left[\frac{2 \cdot L(\mathbf{x})}{3}, A\right]$. Let $\varDelta = A - A' \geq 0$, then

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6} \cdot (x_i - A) + \frac{1}{3}\left|x_i - \frac{A+B}{2}\right| + \frac{1}{2} \cdot (B - x_i), \\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6} \cdot |x_i - A'| + \frac{1}{3}\left|x_i - \frac{A'+B}{2}\right| + \frac{1}{2} \cdot (B - x_i), \\
&\geq \frac{1}{6} \cdot (x_i - A + \varDelta) + \frac{1}{3}\left[\left|x_i - \frac{A+B}{2}\right| - \frac{\varDelta}{2}\right] + \frac{1}{2} \cdot (B - x_i), \\
&= cost(x_i, f(\mathbf{x})).
\end{aligned}$$

**Case 1.2.2.** $x_i' > L(\mathbf{x})$. Let $\varDelta = x_i' - L(\mathbf{x})$. We have $L(\mathbf{x}') = x_i'$. It holds that $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$ when $b(\mathbf{x}) \neq b(\mathbf{x}')$. Now consider the case of $b(\mathbf{x}) = b(\mathbf{x}')$. Let $\varDelta_1 = b(\mathbf{x}) - [L(\mathbf{x}) - l(\mathbf{x}')]$ and $\varDelta_2 = \frac{3}{2} \cdot l(\mathbf{x}') - L(\mathbf{x})$, we have

$$L(\mathbf{x}) + \varDelta_1 - l(\mathbf{x}') = b(\mathbf{x}') = b(\mathbf{x}), L(\mathbf{x}) + \varDelta_2 - l(\mathbf{x}') = \frac{1}{3} \cdot [L(\mathbf{x}) + \varDelta_2].$$

(1) $\varDelta \leq \varDelta_1$. The possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_1, \frac{A_1+B_1}{2}, B_1$ from left to right. It is evident that $b(\mathbf{x}') \geq L(\mathbf{x}) + \varDelta - l(\mathbf{x}')$, we have

$$A_1 = A + \varDelta, B_1 = B + \varDelta.$$

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6} \cdot (x_i - A) + \frac{1}{3}\left|x_i - \frac{A+B}{2}\right| + \frac{1}{2} \cdot (B - x_i) \\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6} \cdot |x_i - A_1| + \frac{1}{3}\left|x_i - \frac{A_1+B_1}{2}\right| + \frac{1}{2} \cdot (B_1 - x_i) \\
&\geq \frac{1}{6} \cdot (x_i - A - \varDelta) + \frac{1}{3}\left[\left|x_i - \frac{A+B}{2}\right| - \varDelta\right] + \frac{1}{2} \cdot (\varDelta + B - x_i) \\
&= cost(x_i, f(\mathbf{x})) - \frac{\varDelta}{6} - \frac{\varDelta}{3} + \frac{\varDelta}{2} \\
&= cost(x_i, f(\mathbf{x}))
\end{aligned}$$

(2) $\varDelta \in (\varDelta_1, \varDelta_2)$. The possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_2, \frac{A_2+B_2}{2}, B_2$ from left to right. It is known that $b(\mathbf{x}') < L(\mathbf{x}) + \varDelta - l(\mathbf{x}') < \frac{1}{3} \cdot [L(\mathbf{x}) + \varDelta]$,

$$A_2 = l(\mathbf{x}') = A + \varDelta_1, B_2 = L(\mathbf{x}) + \varDelta = B + \varDelta.$$

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6} \cdot (x_i - A) + \frac{1}{3}\left|x_i - \frac{A+B}{2}\right| + \frac{1}{2} \cdot (B - x_i) \\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6} \cdot |x_i - A_2| + \frac{1}{3}\left|x_i - \frac{A_2+B_2}{2}\right| + \frac{1}{2} \cdot (B_2 - x_i) \\
&\geq \frac{1}{6} \cdot (x_i - A - \varDelta_1) + \frac{1}{3}\left[\left|x_i - \frac{A+B}{2}\right| - \frac{\varDelta_1 + \varDelta}{2}\right] + \frac{1}{2} \cdot (\varDelta + B - x_i) \\
&= cost(x_i, f(\mathbf{x})) - \frac{\varDelta_1}{6} - \frac{\varDelta_1}{6} - \frac{\varDelta}{6} + \frac{\varDelta}{2} \\
&= cost(x_i, f(\mathbf{x})) - \frac{\varDelta_1}{3} + \frac{\varDelta}{3} \\
&\geq cost(x_i, f(\mathbf{x}))
\end{aligned}$$

(3) $\varDelta \geq \varDelta_2$. The possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_3, \frac{A_3+B_3}{2}, B_3$ from left to right. It is known that $b(\mathbf{x}') < \frac{1}{3} \cdot [L(\mathbf{x}) + \varDelta] \leq L(\mathbf{x}) + \varDelta - l(\mathbf{x}')$, we have

$$A_3 = \frac{2}{3} \cdot [L(\mathbf{x}) + \varDelta] = A + \varDelta_1 + \frac{2}{3} \cdot (\varDelta - \varDelta_2), B_3 = B + \varDelta.$$

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6}\cdot(x_i - A) + \frac{1}{3}\left|x_i - \frac{A+B}{2}\right| + \frac{1}{2}\cdot(B - x_i)\\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6}\cdot|x_i - A_3| + \frac{1}{3}\left|x_i - \frac{A_3+B_3}{2}\right| + \frac{1}{2}\cdot(B_3 - x_i)\\
&\geq \frac{1}{6}\cdot\left\{x_i - A - \left[\Delta_1 + \frac{2\cdot(\Delta-\Delta_2)}{3}\right]\right\}\\
&+ \frac{1}{3}\left[\left|x_i - \frac{A+B}{2}\right| - \frac{\Delta+\Delta_1+\frac{2}{3}\cdot(\Delta-\Delta_2)}{2}\right] + \frac{1}{2}\cdot(\Delta + B - x_i)\\
&= cost(x_i, f(\mathbf{x})) - \frac{\Delta_1}{6} - \frac{\Delta-\Delta_2}{9} - \frac{1}{3}\cdot\frac{\Delta+\Delta_1+\frac{2}{3}\cdot(\Delta-\Delta_2)}{2} + \frac{\Delta}{2}\\
&\geq cost(x_i, f(\mathbf{x})) + \frac{\Delta}{9} + \frac{2}{9}\cdot\Delta_2 - \frac{\Delta_1}{3}\\
&\geq cost(x_i, f(\mathbf{x}))
\end{aligned}$$

**Case 1.3.** $x_i = L(\mathbf{x})$. If $x_i' \leq L(\mathbf{x})$, the facility will move away from $L(\mathbf{x})$, then $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$. If $x_i' > L(\mathbf{x})$, the proof is similar to that of Case 1.2.2.

**Case 2.** $b(\mathbf{x}) \geq L(\mathbf{x}) - l(\mathbf{x})$ and $l(\mathbf{x}) = L(\mathbf{x})$, as shown in Fig. 3(2). Suppose agent $i$ reports her location $x_i$ to $x_i'$.

**Case 2.1.** $x_i \leq b(\mathbf{x})$. It is obvious that $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$.

**Case 2.2.** $x_i = L(\mathbf{x})$.

(1) Only agent $i$ is located at $L(\mathbf{x})$.

If $x_i' \leq L(\mathbf{x})$, $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$. If $x_i' > L(\mathbf{x})$, let $\Delta = x_i' - L(\mathbf{x}) > 0$, we have

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6}\cdot b(\mathbf{x}) + \frac{1}{3}\cdot\frac{b(\mathbf{x})}{2} + \frac{1}{2}\cdot 0\\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6}\cdot|b(\mathbf{x}) - \Delta| + \frac{1}{3}\cdot\left|\frac{b(\mathbf{x})}{2} - \Delta\right| + \frac{\Delta}{2}\\
&\geq cost(x_i, f(\mathbf{x})) - \frac{\Delta}{6} - \frac{\Delta}{3} + \frac{\Delta}{2}\\
&= cost(x_i, f(\mathbf{x}))
\end{aligned}$$

(2) There are multiple agents at location $L(\mathbf{x})$.

If $x_i' \leq L(\mathbf{x})$, The result can only be that the facility is farther from $L(\mathbf{x})$. If $x_i' > L(\mathbf{x})$, the proof is similar to Case 1.2.2 shows agents located at $L(\mathbf{x})$ will not profit.

**Case 3.** $b(\mathbf{x}) < L(\mathbf{x}) - l(\mathbf{x})$ and $l(\mathbf{x}) \leq \frac{2}{3}\cdot L(\mathbf{x})$, as shown in Fig. 3(3). Suppose agent $i$ reports her location $x_i$ to $x_i'$.

**Case 3.1.** $x_i \leq b(\mathbf{x})$. Obviously, $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$.

**Case 3.2.** $x_i \in [l(\mathbf{x}), L(\mathbf{x})]$. If $x_i' \leq L(\mathbf{x})$, $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$. If $x_i' > L(\mathbf{x})$, then $L(\mathbf{x}') = x_i'$. Let $\Delta = L(\mathbf{x}') - L(\mathbf{x}) > 0$. As $\Delta$ increases from zero, let $\Delta$ be $\Delta_1$ when $b(\mathbf{x}') \neq b(\mathbf{x})$ is satisfied for the first time. It is characterized by:

$$\frac{1}{3} \cdot [L(\mathbf{x}) + \Delta_1] = b(\mathbf{x}')$$

Let $\Delta_2$ be the value of $\Delta$ when $b(\mathbf{x}') \geq L(\mathbf{x}') - l(\mathbf{x}')$ is satisfied for the first time after $\Delta \geq \Delta_1$. Let $\Delta_3$ be the value of $\Delta$ when $b(\mathbf{x}') \leq L(\mathbf{x}') - l(\mathbf{x}')$ is satisfied for the first time after $\Delta \geq \Delta_2$. Let $\Delta_4$ be the value of $\Delta$ when $L(\mathbf{x}') - l(\mathbf{x}') = \frac{L(\mathbf{x}')}{3}$ is satisfied for the first time after $\Delta \geq \Delta_3$.

(1) $\Delta < \Delta_1$. If $x_i = l(\mathbf{x})$, it is obvious the possible facility is farther from $\frac{2 \cdot L(\mathbf{x})}{2}$, agents will not lie. If $x_i \neq l(\mathbf{x})$, the possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_1, \frac{A_1+B_1}{2}, B_1$ from left to right. It is evident that $b(\mathbf{x}') = b(\mathbf{x}) < x_i' - l(\mathbf{x}')$, and we have

$$A_1 = A + \frac{2}{3} \cdot \Delta, B_1 = B + \Delta$$

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6} \cdot |x_i - A| + \frac{1}{3} \cdot \left|x_i - \frac{A+B}{2}\right| + \frac{1}{2} \cdot (B - x_i) \\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6} \cdot |x_i - A_1| + \frac{1}{3} \cdot \left|x_i - \frac{A_1+B_1}{2}\right| + \frac{1}{2} \cdot (B_1 - x_i) \\
&= \frac{1}{6} \cdot \left|x_i - \left(A + \frac{2}{3} \cdot \Delta\right)\right| + \frac{1}{3} \cdot \left|x_i - \left(\frac{A+B}{2} + \frac{5}{6} \cdot \Delta\right)\right| + \frac{1}{2} \cdot (B + \Delta - x_i) \\
&\geq cost(x_i, f(\mathbf{x})) - \frac{1}{9} \cdot \Delta - \frac{5}{18} \cdot \Delta + \frac{1}{2} \cdot \Delta \\
&\geq cost(x_i, f(\mathbf{x}))
\end{aligned}$$

(2) $\Delta \in [\Delta_1, \Delta_2)$. The possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_2, \frac{A_2+B_2}{2}, B_2$ from left to right. It is evident that $b(\mathbf{x}') \leq L(\mathbf{x}') - l(\mathbf{x}')$, and we have

$$A_2 = A + \frac{2}{3} \cdot \Delta, B_2 = B + \Delta.$$

According to the proof in (1), her cost will not decrease.

(3) $\Delta \in [\Delta_2, \Delta_3)$, where $b(\mathbf{x}') \geq L(\mathbf{x}') - l(\mathbf{x}')$. Let $\delta_1 = \Delta - \Delta_2 \geq 0$ and the possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_3, \frac{A_3+B_3}{2}, B_1$ from left to right. We have

$$A_3 = A + \frac{2}{3} \cdot \Delta_2 + \delta_1, B_3 = B + \Delta_2 + \delta_1.$$

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6} \cdot |x_i - A| + \frac{1}{3} \cdot \left|x_i - \frac{A+B}{2}\right| + \frac{1}{2} \cdot (B - x_i), \\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6} \cdot |x_i - A_3| + \frac{1}{3} \cdot \left|x_i - \frac{A_3+B_3}{2}\right| + \frac{1}{2} \cdot (B_3 - x_i) \\
&= \frac{1}{6} \cdot \left|x_i - \left(A + \frac{2}{3} \cdot \Delta_2 + \delta_1\right)\right| + \frac{1}{3} \cdot \left|x_i - \left(\frac{A+B}{2} + \frac{5}{6} \cdot \Delta_2 + \delta_1\right)\right| \\
&\quad + \frac{1}{2} \cdot (B + \Delta_2 + \delta_1 - x_i) \\
&\geq cost(x_i, f(\mathbf{x})) - \frac{1}{9} \cdot \Delta_2 - \frac{5}{18} \cdot \Delta_2 + \frac{1}{2} \cdot \Delta_2 - \frac{\delta_1}{6} - \frac{\delta_1}{3} + \frac{\delta_1}{2} \\
&\geq cost(x_i, f(\mathbf{x}))
\end{aligned}$$

(4) $\Delta \in [\Delta_3, \Delta_4)$, where $b(\mathbf{x}') \leq L(\mathbf{x}') - l(\mathbf{x}')$ and $l(\mathbf{x}') \geq \frac{2 \cdot L(\mathbf{x}')}{3}$. Let $\delta_2 = \Delta - \Delta_3 \geq 0$ and The possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_4$, $\frac{A_4+B_4}{2}$, $B_4$ from left to right. We have

$$A_4 = A + \frac{2}{3} \cdot \Delta_2 + \Delta_3 - \Delta_2 = A - \frac{\Delta_2}{3} + \Delta_3, B_3 = B + \Delta_3 + \delta_2.$$

$$\begin{aligned}
cost(x_i, f(\mathbf{x})) &= \frac{1}{6} \cdot |x_i - A| + \frac{1}{3} \cdot \left| x_i - \frac{A+B}{2} \right| + \frac{1}{2} \cdot (B - x_i) \\
cost(x_i, f(x_i', \mathbf{x}_{-i})) &= \frac{1}{6} \cdot |x_i - A_4| + \frac{1}{3} \cdot \left| x_i - \frac{A_4+B_4}{2} \right| + \frac{1}{2} \cdot (B_4 - x_i) \\
&= \frac{1}{6} \cdot \left| x_i - \left( A - \frac{\Delta_2}{3} + \Delta_3 \right) \right| + \frac{1}{3} \cdot \left| x_i - \left( \frac{A+B}{2} + \Delta_3 - \frac{2 \cdot \Delta_2}{3} + \frac{\delta_2}{2} \right) \right| \\
&\quad + \frac{1}{2} \cdot (B + \Delta_3 + \delta_2 - x_i) \\
&\geq cost(x_i, f(\mathbf{x})) + \frac{5 \cdot \Delta_2}{18} + \frac{\delta_2}{3} \\
&\geq cost(x_i, f(\mathbf{x}))
\end{aligned}$$

(5) $\Delta \geq \Delta_4$. As $\Delta$ increases gradually from $\Delta_4$, $L(\mathbf{x}') - l(\mathbf{x}') > \frac{L(\mathbf{x}')}{3}$ is always satisfied before $b(\mathbf{x}')$ changes. The possible locations of $f(x_i', \mathbf{x}_{-i})$ are denoted as $A_5$, $\frac{A_5+B_5}{2}$, $B_5$ from left to right. We have

$$A_5 = A + \frac{2}{3} \cdot \Delta, B_5 = B + \Delta.$$

The proof is similar to Case 3.2(1).

When $\Delta$ continues to increase until $b(\mathbf{x}')$ changes for the first time, the output facility location is far enough away from $x_i$, and the cost of agent $i$ will not be reduced.

**Case 4.** $b(\mathbf{x}) < L(\mathbf{x}) - l(\mathbf{x})$ and $l(\mathbf{x}) > \frac{2}{3} \cdot L(\mathbf{x})$. If $l(\mathbf{x}) = L(\mathbf{x})$. It is easy to prove any agents can not benefit by misreporting their location; otherwise, as shown in Fig. 3(4). Suppose agent $i$ reports her location $x_i$ to $x_i'$.

**Case 4.1.** $x_i \leq b(\mathbf{x})$. Obviously, $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$.

**Case 4.2.** $x_i \in \big[l(\mathbf{x}), L(\mathbf{x})\big)$. If $x_i' \leq L(\mathbf{x})$. The proof is similar to Case 1.2.1. If $x_i' = L(\mathbf{x}) + \Delta > L(\mathbf{x})$, it is clear that $cost(x_i, f(\mathbf{x})) \leq cost(x_i, f(x_i', \mathbf{x}_{-i}))$ when $b(\mathbf{x}) \neq b(\mathbf{x}')$. If $b(\mathbf{x}) = b(\mathbf{x}')$, as $\Delta$ increases from 0, the proof is similar to Case 1.2.2.

**Case 4.3.** $x_i = L(\mathbf{x})$. If $x_i' \leq L(\mathbf{x})$, the facility will move away from $L(\mathbf{x})$ and agent $i$ can not benefit. If $x_i' > L(\mathbf{x})$, the proof is similar to the second part of Case 4.2.

In conclusion, Mechanism 3 is a strategy-proof mechanism.

**Approximation Ratio.** Given any instance $\mathbf{x} = (x_1, x_2, ..., x_n)$, our analysis proceeds as follows.

**Case 1.** $b(\mathbf{x}) \geq L(\mathbf{x}) - l(\mathbf{x})$, as shown in Fig. 3(1) or Fig. 3(2).

In this case, we have $MC(\mathbf{x}, OPT) = b(\mathbf{x})$. For each output of $f$, the maximum cost is less than $b(\mathbf{x})$, then it holds that

$$\frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} \leq \frac{b(\mathbf{x})}{MC(\mathbf{x}, OPT)} = 1$$

**Case 2.** $b(\mathbf{x}) < L(\mathbf{x}) - l(\mathbf{x})$ and $l(\mathbf{x}) \leq \frac{2}{3} \cdot L(\mathbf{x})$.

Let $l(\mathbf{x}) = \frac{L(\mathbf{x})}{3} + \Delta \leq \frac{2L(\mathbf{x})}{3}, \Delta \in (0, \frac{L(\mathbf{x})}{3}]$. We have

$$MC(\mathbf{x}, OPT) = \max\left\{b(\mathbf{x}), \frac{L(\mathbf{x}) - l(\mathbf{x})}{2}\right\} = \max\left\{b(\mathbf{x}), \frac{L(\mathbf{x})}{3} - \frac{\Delta}{2}\right\}.$$

$$\begin{aligned} MC(\mathbf{x}, f) &\leq \frac{1}{6} \cdot \frac{L(\mathbf{x})}{3} + \frac{1}{3} \cdot \max\left\{b(\mathbf{x}), cost(l(\mathbf{x}), \frac{5L(\mathbf{x})}{6})\right\} + \frac{1}{2} \cdot (L(\mathbf{x}) - l(\mathbf{x})) \\ &= \frac{L(\mathbf{x})}{18} + \frac{1}{3} \cdot \max\left\{b(\mathbf{x}), \frac{L(\mathbf{x})}{2} - \Delta\right\} + \frac{L(\mathbf{x})}{3} - \frac{\Delta}{2} \end{aligned}$$

**Case 2.1.** $b(\mathbf{x}) \geq \frac{L(\mathbf{x})}{2} - \Delta \in \left[\frac{L(\mathbf{x})}{6}, \frac{L(\mathbf{x})}{2}\right)$.

$$\begin{aligned} \frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} &= \frac{\frac{L(\mathbf{x})}{18} + \frac{b(\mathbf{x})}{3} + \frac{L(\mathbf{x})}{3} - \frac{\Delta}{2}}{b(\mathbf{x})} \leq \frac{1}{3} + \frac{\frac{7L(\mathbf{x})}{18} + \frac{b(\mathbf{x})}{2} - \frac{L(\mathbf{x})}{4}}{b(\mathbf{x})} \\ &= \frac{1}{3} + \frac{14L(\mathbf{x}) + 18b(\mathbf{x}) - 9L(\mathbf{x})}{36b(\mathbf{x})} \leq \frac{1}{3} + \frac{1}{2} + \frac{5}{6} = \frac{5}{3} \end{aligned}$$

**Case 2.2.** $b(\mathbf{x}) < \frac{L(\mathbf{x})}{2} - \Delta$.

$$\frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} \leq \frac{\frac{L(\mathbf{x})}{18} + \frac{1}{3} \cdot \left(\frac{L(\mathbf{x})}{2} - \Delta\right) + \frac{L(\mathbf{x})}{3} - \frac{\Delta}{2}}{\frac{L(\mathbf{x})}{3} - \frac{\Delta}{2}} = \frac{10L(\mathbf{x}) - 15\Delta}{6L(\mathbf{x}) - 9\Delta} = \frac{5}{3}.$$

**Case 3.** $b(\mathbf{x}) < L(\mathbf{x}) - l(\mathbf{x})$ and $l(\mathbf{x}) > \frac{2}{3} \cdot L(\mathbf{x})$. We have

$$\begin{aligned} MC(\mathbf{x}, OPT) &= \max\left\{b(\mathbf{x}), \frac{L(\mathbf{x}) - l(\mathbf{x})}{2}\right\}, \\ MC(\mathbf{x}, f) &= \frac{1}{6} \cdot \max\left\{b(\mathbf{x}), L(\mathbf{x}) - l(\mathbf{x})\right\} + \frac{1}{3} \cdot MC(\mathbf{x}, OPT) \\ &\quad + \frac{1}{2} \cdot \max\left\{b(\mathbf{x}), L(\mathbf{x}) - l(\mathbf{x})\right\}. \\ \frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} &\leq \frac{2}{6} + \frac{1}{3} + \frac{1}{2} = \frac{5}{3} \end{aligned}$$

**Tight Example.** Consider an instance $\mathbf{x} = (4, 6)$. It is clear that $OPT(\mathbf{x}) = 5$ and the optimal value is 1. Mechanism 3 outputs 4, 5 and 6 with probabilities $\frac{1}{6}$, $\frac{1}{3}$ and $\frac{1}{2}$, respectively. Therefore,

$$\frac{MC(\mathbf{x}, f)}{MC(\mathbf{x}, OPT)} = \frac{5}{3}$$

## References

1. Alon, N., Feldman, M., Procaccia, A.D., Tennenholtz, M.: Strategyproof approximation mechanisms for location on networks. CoRR, abs/0907.2049 (2009)
2. Chan, H., Filos-Ratsikas, A., Li, B., Li, M., Wang, C.: Mechanism design for facility location problems: a survey. In: 30th International Joint Conference on Artificial Intelligence, pp. 4356–4365. International Joint Conferences on Artificial Intelligence Organization (2021)
3. Chan, H., Fu, X., Li, M., Wang, C.: Mechanism design for reducing agent distances to prelocated facilities. In: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, pp. 2180–2182 (2024)
4. Chan, H., Wang, C.: Mechanism design for improving accessibility to public facilities. In: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, pp. 2116–2124 (2023)
5. Chen, X., Hu, X., Tang, Z., Wang, C.: Tight efficiency lower bounds for strategy-proof mechanisms in two-opposite-facility location game. Inf. Process. Lett. **168**, 106098 (2021)
6. Cheng, Y., Yu, W., Zhang, G.: Strategy-proof approximation mechanisms for an obnoxious facility game on networks. Theoret. Comput. Sci. **497**, 154–163 (2013)
7. Deligkas, A., Filos-Ratsikas, A., Voudouris, A.A.: Heterogeneous facility location with limited resources. Games Econom. Behav. **139**, 200–215 (2023)
8. Feldman, M., Wilf, Y.: Strategyproof facility location and the least squares objective. In: Proceedings of the Fourteenth ACM Conference on Electronic Commerce, pp. 873–890 (2013)
9. Fong, C.K.K., Li, M., Lu, P., Todo, T., Yokoo, M.: Facility location games with fractional preferences. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
10. Fotakis, D., Tzamos, C.: Strategyproof facility location for concave cost functions. In: Proceedings of the Fourteenth ACM Conference on Electronic Commerce, pp. 435–452 (2013)
11. Fotakis, D., Tzamos, C.: On the power of deterministic mechanisms for facility location games. ACM Trans. Econ. Comput. (TEAC) **2**(4), 1–37 (2014)
12. Li, M., Lu, P., Yao, Y., Zhang, J.: Strategyproof mechanism for two heterogeneous facilities with constant approximation ratio. In: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, pp. 238–245 (2021)
13. Lu, P., Sun, X., Wang, Y., Zhu, Z.A.: Asymptotically optimal strategy-proof mechanisms for two-facility games. In: Proceedings of the 11th ACM Conference on Electronic Commerce, pp. 315–324 (2010)
14. Procaccia, A.D., Tennenholtz, M.: Approximate mechanism design without money. ACM Trans. Econ. Comput. (TEAC) **1**(4), 1–26 (2013)
15. Qin, Z., Chan, H., Wang, C., Zhang, Y.: Mechanism design for building optimal bridges between regions. In: Annual Conference on Theory and Applications of Models of Computation, pp. 332–343. Springer, Singapore (2024)
16. Serafino, P., Ventre, C.: Truthful mechanisms without money for non-utilitarian heterogeneous facility location. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29 (2015)
17. Serafino, P., Ventre, C.: Heterogeneous facility location without money. Theoret. Comput. Sci. **636**, 27–46 (2016)

18. Tang, Z., Wang, C., Zhang, M., Zhao, Y.: Mechanism design for facility location games with candidate locations. In: Wu, W., Zhang, Z. (eds.) COCOA 2020. LNCS, vol. 12577, pp. 440–452. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64843-5_30
19. Xu, X., Li, B., Li, M., Duan, L.: Two-facility location games with minimum distance requirement. J. Artif. Intell. Res. **70**, 719–756 (2021)
20. Ye, D., Mei, L., Zhang, Y.: Strategy-proof mechanism for obnoxious facility location on a line. In: Xu, D., Du, D., Du, D. (eds.) COCOON 2015. LNCS, vol. 9198, pp. 45–56. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21398-9_4
21. Yuan, H., Wang, K., Fong, K.C., Zhang, Y., Li, M.: Facility location games with optional preference. In: ECAI 2016, pp. 1520–1527. IOS Press, Amsterdam (2016)
22. Zou, S., Li, M.: Facility location games with dual preference. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, pp. 615–623 (2015)

# Computing Approximate Mixed Nash Equilibria for Symmetric Weighted Congestion Games

Chunying Ren[1], Zijun Wu[2](✉), Xiaoguang Yang[3], and Guoqing Zhang[4]

[1] Center for Combinatorics, Nankai University, Tianjin 300071, China
`Rcy9820230019@nankai.edu.cn`

[2] School of Automotive and Transportation Engineering, Hefei University of Technology, Anhui 230009, China
`zijunwu1984a@163.com`

[3] Academy of Mathematics and System Science, Chinese Academy of Sciences, Beijing 100190, China
`xgyang@iss.ac.cn`

[4] Supply Chain and Logistics Optimization Research Center, Department of Mechanical, Automotive and Materials Engineering, University of Windsor, Windsor, ON N9B 3P4, Canada
`gzhang@uwindsor.ca`

**Abstract.** We concern the computation of approximate mixed Nash equilibria in symmetric weighted congestion games, which has been shown to be PPAD-complete. We focus our discussion only on affine linear latency functions, and propose an algorithm deriving from the best response dynamics. Our algorithm efficiently computes an $\epsilon$-approximate mixed Nash equilibrium within a polynomial runtime parameterized mainly by the maximum player weight $W$, where $\epsilon \in (0, 1)$ is an arbitrary small constant. This then provides the first polynomial runtime algorithm for computing an $\epsilon$-approximate mixed Nash equilibrium in a weighted congestion game, though the players are programmed to have the same strategy set, the latency functions are assumed to be affine linear, and the polynomial runtime is still parametric.

**Keywords:** Weighted congestion game · Symmetric · Affine linear latency function · Potential function · Best response dynamic

## 1 Introduction

Design efficient algorithms for computing Nash equilibria of *congestion games* ([1] and [2]) is an important research topic of *algorithmic game theory* [3,4]. While various fundamental properties of Nash equilibria in congestion games have been obtained in recent years, see, e.g., [5–10], this problem is still open, due to its extreme computational complexity.

Congestion games are non-cooperative games, in which selfish players compete for finite sets of resources. We focus only on *weighted* congestion games. A prototype is a transportation network [11], in which players want to deliver certain unsplitable commodities by quickest paths between specific origin-destination pairs. Here, the arcs of the network are the resources, and the transportation demands of these commodities are the weights of the players. As the demands can not split, these games are essentially finite games, and so admit *mixed Nash equilibria*, see [12].

While weighted congestion games have much more explicit structure, the computation of their mixed Nash equilibria is actually not easier than arbitrary finite games. In fact, finding a mixed Nash equilibrium for a two-player game has already been *PPAD-complete*, see [13]. Here, PPAD shorts for the term "Polynomial Parity Argument on Directed Graphs", which is a complexity class introduced by [14]. So far, we only know that PPAD lies between FP and FNP. As whether FP $\neq$ FNP or not is unknown, we still do not have a polynomial time exact algorithm for computing a mixed Nash equilibrium.

Due to the PPAD-completeness, we will not discuss an arbitrary symmetric weighted congestion game, but these with affine linear latency functions, and with players' weights value in a bounded interval $[1, W]$ for an arbitrary constant $W \geq 1$. With such setting, we propose a simply algorithm based on *best response dynamics*, which computes an *$\epsilon$-approximate mixed Nash equilibrium* within a polynomial runtime parameterized by the constant $W$.

## 1.1 Our Contribution

We extend best-response dynamics to compute mixed Nash equilibria for symmetric weighted congestion games with affine linear latency functions. To apply best response dynamics, we first discretize the mixed strategy space of each player with a suitable parameter $\delta \in (0, 1)$, see Algorithm 2, which ensures that the computation of a best response strategy of a player in the resulting discrete strategy space has a polynomial time of $\frac{1}{\delta}$. This combining with the best response dynamics then yields an $\epsilon$-approximate mixed Nash equilibrium algorithm for weighted congestion games, see Algorithm 1.

To analyze the runtime of our algorithm, we generalize the *Rosenthal's potential function* [15] of pure strategy profiles to that of mixed strategy profiles $\pi$ for the symmetric weighted congestion with affine linear latency functions, which then gives the time complexity of Algorithm 1. We prove that Algorithm 1 outputs an $\epsilon$-approximate mixed Nash equilibrium within $O(\frac{N \cdot (W + d! \cdot W^{d+1})}{\epsilon \cdot \delta} \cdot \log(N \cdot C_{\max}))$ iterations, where $N$ is the number of players and $C_{\max}$ is the players' maximum expected (latency) cost value. See Theorem 2 for a detailed result. Due to lack of space, we defer all proofs to the full version of the paper.

## 1.2 Related Work

The existence of Nash equilibria in congestion games was obtained in e.g., [1,12,16,17], and others. [1] showed that every *unweighted* congestion game admits pure Nash equilibria as it has a potential function, i.e., it is a *potential game*. [16] proved that weighted congestion games with affine linear latency functions are also potential games, and so admit pure Nash equilibria. Besides, [17] proved that it is NP-hard to decide if an arbitrary weighted congestion game has a pure Nash equilibrium. Since weighted congestion games are finite games, mixed Nash equilibria always exist, see [12]. However, computing equilibria for congestion games is a very tough problem. [13] proved that computing a mixed Nash equilibrium for a two-player game is *PPAD-complete* [14]. The computational complexity significantly questions the importance of Nash equilibria as a solution concept for the behavior of rational players. This then leads to the concept of *approximate Nash equilibria*, which replace exact Nash equilibria as asymptotically stable solutions.

[18] proved that the best-response dynamics converge to $\frac{1}{1-\epsilon}$-approximate pure Nash equilibria within finite iterations for symmetric unweighted congestion games with latency functions fulfilling the so-called $\alpha$ *bounded jump conditions*, where $\epsilon \geq 0$ is an arbitrary small constant. [19] proposed the first algorithm for computing approximate pure Nash equilibria in unweighted congestion games with polynomial latency functions of degree at most an integer $d \geq 1$. They designed an algorithm that converges to a $d^{O(d)}$-approximate pure Nash equilibrium in a (parameterized) polynomial time. Then [20] generalized this result to weighted congestion games, and obtained an efficient $d^{O(d^2)}$-approximate pure Nash equilibrium algorithm. Moreover, [21] designed a random algorithm based on the work of [20], which produces a $d^{d+o(d)}$-approximate pure Nash equilibrium with a high probability. Along this direction, [22] gave the current state-of-the-art results. They designed an efficient deterministic algorithm, which computes a $\frac{\rho}{1-\epsilon}$-approximate pure Nash equilibrium within a finite number of iterations, where $\rho = \frac{2 \cdot W \cdot (d+1)}{2 \cdot W + d + 1}$.

As a generalization of [22], we focus on the computation of approximate mixed Nash equilibria for symmetric weighted congestion games, and propose an $\epsilon$-approximate mixed Nash equilibrium algorithm, though under the constraints of affine linear latency functions.

### 1.3 Outline of the Paper

The remaining of this paper is organized as follows. Section 2 defines the model. Then we propose a potential function on mixed profile space, and design and analyze an $\epsilon$-approximate mixed Nash equilibrium algorithm in Sect. 3. Finally, a short summary is given in Sect. 4.

## 2 Model and Preliminaries

### 2.1 Weighted Congestion Games

Formally, a *weighted congestion game* $\Gamma$ is written as a tuple $(\mathcal{N}, \mathcal{E}, (\Sigma_u)_{u \in \mathcal{N}}, (c_e)_{e \in \mathcal{E}}, (w_u)_{u \in \mathcal{N}})$. Here, $\mathcal{N} = \{1, \ldots, N\}$ is a set of $N$ players, and $\mathcal{E} =$

$\{1, \ldots, E\}$ is a set of $E$ resources. Each player $u \in \mathcal{N}$ has a traffic weight $w_u \geq 0$ and a strategy set $\Sigma_u \subseteq 2^{\mathcal{E}}$. To facilitate our discussion, we assume, w.l.o.g., that $w_u \geq 1$ for all $u \in \mathcal{N}$. Note that $\Gamma$ is *unweighted* when $w_u = w_v$, and $\Gamma$ is *symmeric* when $\Sigma_u = \Sigma_v$, for all $u, v \in \mathcal{N}$. For each resource $e$, $c_e : \mathbb{R}_+ \to \mathbb{R}_+$ denotes a non-decreasing (latency) cost function.

Every pure strategy of a player $u \in \mathcal{N}$ induces a pure strategy subprofile $f_u = (f_{s_u})_{s_u \in \Sigma_u}$ fulfilling condition (2.1) below,

$$\begin{aligned} &f_{s_u} = w_u \cdot \mathbf{1}(f_u, s_u), \ \forall s_u \in \Sigma_u, \\ &\sum_{s_u \in \Sigma_u} \mathbf{1}(f_u, s_u) = 1, \end{aligned} \tag{2.1}$$

where $\mathbf{1}(f_u, s_u)$ is a $\{0, 1\}$-valued function, which indicates whether strategy $s_u$ is used by player $u$, i.e., $\mathbf{1}(f_u, s_u) = 1$, or not, i.e., $\mathbf{1}(f_u, s_u) = 0$. We denote by $\mathcal{F}_u$ the set of all pure strategy subprofiles of player $u \in \mathcal{N}$ fulfilling condition (2.1). A *pure strategy profile* (simply, *pure profile*) $f$ of all players is then a vector $(f_u)_{u \in \mathcal{N}} \in \prod_{u \in \mathcal{N}} \mathcal{F}_u$.

A *mixed strategy* of player $u \in \mathcal{N}$ is a probability distribution $\pi_u = (\pi_{u,s_u})_{s_u \in \Sigma_u}$ over $\mathcal{F}_u$, with which player $u$ draws a *random* strategy $s_u$ from $\Sigma_u$ and then sends its weight $w_u$ along this random strategy. Hence, each mixed strategy $\pi_u$ induces in a random subprofile $\mathbf{f}_u = (\mathbf{f}_{s_u})_{s_u \in \Sigma_u}$ fulfilling condition (2.2) below,

$$\begin{aligned} &\mathbf{f}_{s_u} = w_u \cdot \mathbf{1}(\mathbf{f}_u, s_u), && \forall s_u \in \Sigma_u, \\ &\sum_{s_u \in \Sigma_u} \mathbf{1}(\mathbf{f}_u, s_u) = 1, \\ &\mathbf{P}[\mathbf{f}_{s_u} = w_u] = \pi_{u,s_u}, \ \mathbf{E}[\mathbf{f}_{s_u}] = w_u \cdot \pi_{u,s_u} \text{ and} \\ &\mathbf{Var}[\mathbf{f}_{s_u}] = w_u^2 \cdot \pi_{u,s_u} \cdot (1 - \pi_{u,s_u}) && \forall s_u \in \Sigma_u. \end{aligned} \tag{2.2}$$

Clearly, $\mathbf{f}_u$ has $\mathcal{F}_u$ as its *state space.*

A *mixed strategy profile* (simply, *mixed profile*) is then the product $\pi = (\pi_u)_{u \in \mathcal{N}}$ of mixed strategies of all players, i.e., players draw their random strategies mutually independently. Then $\mathbf{f} = (\mathbf{f}_u)_{u \in \mathcal{N}}$ is the random strategy profile (simply, random profile) of $\pi$, and $\mathbf{f}_e = \sum_{s_u \in \cup_{u \in \mathcal{N}} \Sigma_u:\ e \in s_u} \mathbf{f}_{s_u}$ is the random weight value of resource $e \in \mathcal{E}$. Hence, player $u \in \mathcal{N}$ has an *expected (latency) cost* of

$$\begin{aligned} C_u(\pi) = \mathbf{E}[C_u(\mathbf{f})] &= \mathbf{E}[\sum_{s_u \in \Sigma_u} \mathbf{f}_{s_u} \cdot c_{s_u}(\mathbf{f})] \\ &= \sum_{s_u \in \Sigma_u} w_u \cdot \mathbf{E}[\mathbf{1}(\mathbf{f}_u, s_u) \cdot \sum_{e \in \mathcal{E}:\ e \in s_u} c_e(\mathbf{f}_e)] \\ &= \sum_{e \in \mathcal{E}} w_u \cdot \mathbf{E}[\mathbf{1}(\mathbf{f}_u, e) \cdot c_e(\mathbf{f}_e)] = \sum_{e \in \mathcal{E}} \mathbf{E}[\mathbf{f}_{e|u} \cdot c_e(\mathbf{f}_e)], \end{aligned} \tag{2.3}$$

w.r.t. a mixed profile $\pi$. Similar to $f_{e|u}$, $\mathbf{f}_{e|u} := \sum_{s_u \in \Sigma_u:\ e \in s_u} \mathbf{f}_{s_u}$ is the restriction of random resource weight $\mathbf{f}_e$ to player $u$.

In the sequel, we view pure strategies and profiles as particular mixed strategies and profiles with a variance of zero. Moreover, we do not distinguish a random profile $\mathbf{f}$ with its mixed profile $\pi$, unless there is an ambiguity.

### 2.2 Equilibria

A *pure Nash equilibrium* (PNE) is a pure profile $\tilde{f} = (\tilde{f}_u)_{u \in \mathcal{N}}$ satisfying condition (2.4) below,

$$C_u(\tilde{f}_u, \tilde{f}_{-u}) \leq C_u(f_u, \tilde{f}_{-u}), \quad \forall f_u \in \mathcal{F}_u,\ \forall u \in \mathcal{N}. \tag{2.4}$$

Clearly, condition (2.4) means that a unilateral change of pure strategy cannot decrease the cost in a PNE profile.

Similarly, a mixed profile $\tilde{\pi}$ is a *mixed Nash equilibrium* (MINE), if it fulfills the following condition (2.5),

$$C_u(\tilde{\pi}) = C_u(\tilde{\pi}_u, \tilde{\pi}_{-u}) = \mathbf{E}[C_u(\tilde{\mathbf{f}}_u, \tilde{\mathbf{f}}_{-u})] \leq C_u(s_u, \tilde{\pi}_{-u}) \quad \forall s_u \in \Sigma_u,\ \forall u \in \mathcal{N}, \tag{2.5}$$

where $\tilde{\mathbf{f}}_u$ and $\tilde{\mathbf{f}}_{-u}$ are random subprofiles induced by mixed strategy $\tilde{\pi}_u$ of player $u$ and by mixed subprofile $\tilde{\pi}_{-u}$ of opponents of player $u$, respectively, and

$$C_u(s_u, \tilde{\pi}_{-u}) := \mathbf{E}[C_u(f_u, \tilde{\mathbf{f}}_{-u})] \tag{2.6}$$

is the expected (latency) cost of player $u$ when it uses pure strategy $s_u$ and the others follow the mixed subprofile $\tilde{\pi}_{-u}$, and $f_u$ is just the pure subprofile induced by pure strategy $s_u$.

Clearly, $C_u(s_u, \tilde{\pi}_{-u})$ defined in (2.6) is the expected (latency) cost of strategy $s_u \in \Sigma_u$ when the other players follow the mixed subprofile $\tilde{\pi}_{-u}$. Hence, definition (2.5) shows that mixed strategy $\tilde{\pi}_u = (\tilde{\pi}_{u,s_u})_{s_u \in \Sigma_u}$ is a *best-response* to the mixed subprofile $\tilde{\pi}_{-u} = (\tilde{\pi}_{u'})_{u' \in \mathcal{N} \setminus \{u\}} = (\tilde{\pi}_{u',s_{u'}})_{s_{u'} \in \Sigma_{u'}, u' \in \mathcal{N} \setminus \{u\}}$ for each $u \in \mathcal{N}$, since $\tilde{\pi}_u$ concentrates only on strategies with minimum expected cost w.r.t. mixed subprofile $\tilde{\pi}_{-u}$, i.e., its support set includes only strategies with minimum expected (latency) cost.

## 3 Computing $\epsilon$-Approximate Mixed Nash Equilibria

We focus on weighted congestion games with affine linear latency functions fulfilling Condition 1 below.

**Condition 1.** *Each (latency) cost function $c_e : [0, \infty) \rightarrow [0, \infty)$ is affine linear, and has a form of $c_e(x_e) = \alpha_e \cdot x_e + \beta_e$ for any $x_e \in [0, \infty)$ and $e \in \mathcal{E}$, where $a_e \geq 1$ and $\beta_e \geq 1$. Let $\alpha_{max} = max_{e \in \mathcal{E}} \alpha_e$ and $\beta_{max} = max_{e \in \mathcal{E}} \beta_e$ be the maximum values of slope and intercept, respectively.*

Formally, we call a function $\Phi$ mapping each mixed profile $\pi$ to a real value $\Phi(\pi)$ a *b-potential function of mixed profiles* for a constant $b > 0$ if

$$\Phi(\pi) - \Phi(\pi'_u, \pi_{-u}) = \Phi(\pi_u, \pi_{-u}) - \Phi(\pi'_u, \pi_{-u}) = b \cdot [C_u(\pi_u, \pi_{-u}) - C_u(\pi'_u, \pi_{-u})]$$

for each mixed profile $\pi$, each player $u$, and each mixed strategy $\pi'_u$ of player $u$. We show below that weighted congestion games fulfilling Condition 1 have a 2-potential function.

## 3.1 Potential Functions

Given an arbitrary weighted congestion game $\Gamma$ fulfilling Condition 1. We now consider an arbitrary player $u \in \mathcal{N}$, an arbitrary mixed strategy $\pi_u = (\pi_{u,s_u})_{s_u \in \Sigma_u}$ of player $u$, and an arbitrary mixed subprofile $\pi_{-u}$ of his opponents. We denote by $\pi_{u,e} := \sum_{s_u \in \Sigma_u:\ e \in s_u} \pi_{u,s_u}$ the probability of random event that player $u$ uses resource $e$, i.e., $\pi_{u,e} = \mathbf{P}[\mathbf{1}(\mathbf{f}_u, e) = 1]$, and by $\mathcal{A}(\pi_u) := (\pi_{u,e})_{e \in \mathcal{E}}$ the resulting probability vector on resources. Let $\mathbf{f}$ be the random profile induced by mixed profile $\pi = (\pi_u, \pi_{-u})$.

As resource (latency) cost is affine linear, i.e. satisfying Condition 1, then player $u$ has a (latency) cost of $C_u(\pi) = \sum_{e \in \mathcal{E}} \mathbf{E}[\mathbf{f}_{e|u} \cdot c_e(\mathbf{f}_e)] = \sum_{e \in \mathcal{E}} w_u \cdot \pi_{u,e} \cdot c_{u,e}(\pi_{-u,e})$. Here, $\pi_{-u,e} := (\pi_{u',e})_{u' \in \mathcal{N} \setminus \{u\}}$, and $c_{u,e}(\pi_{-u,e})$ is a player-dependent (latency) cost for resource $e \in \mathcal{E}$, which is defined as

$$\begin{aligned} c_{u,e}(\pi_{-u,e}) &= \mathbf{E}[c_e(\mathbf{f}_e) \mid \mathbf{1}(\mathbf{f}_u, e) = 1] = \alpha_e \cdot (w_u + \sum_{u' \in \mathcal{N} \setminus \{u\}} \mathbf{E}[\mathbf{f}_{e|u'}]) + \beta_e \\ &= \alpha_e \cdot (w_u + \sum_{u' \in \mathcal{N} \setminus \{u\}} \pi_{u',e} \cdot w_{u'}) + \beta_e. \end{aligned} \tag{3.1}$$

Clearly, the player-dependent (latency) cost $c_{u,e}(\pi_{-u,e})$ is the expected (latency) cost of resource $e$ conditioned on the event that player $u$ uses resource $e$.

This, combined with inequality (2.5), yield that a mixed profile $\tilde{\pi}$ is a MINE if and only if for each player $u$, and each strategy $s_u \in \Sigma_u$,

$$\sum_{e \in \mathcal{E}} \tilde{\pi}_{u,e} \cdot c_{u,e}(\tilde{\pi}_{-u,e}) = \sum_{s'_u \in \Sigma_u} \tilde{\pi}_{u,s'_u} \cdot \sum_{e \in s'_u} c_{u,e}(\tilde{\pi}_{-u,e}) \leq \sum_{e \in s_u} c_{u,e}(\tilde{\pi}_{-u,e}), \tag{3.2}$$

which is, in turn, equivalent to the condition that

$$\sum_{e \in \mathcal{E}} \tilde{\pi}_{u,e} \cdot c_{u,e}(\tilde{\pi}_{-u,e}) \leq \sum_{e \in \mathcal{E}} \pi'_{u,e} \cdot c_{u,e}(\tilde{\pi}_{-u,e}) \tag{3.3}$$

for an arbitrary mixed strategy $\pi'_u$ of player $u$ with $\pi'_{u,e}$ for all $e \in \mathcal{E}$ and $u \in \mathcal{N}$.

For pure strategies of weighted congestion games satisfying Condition 1, [15] constructed a *Rosenthal's potential function* defined in (3.4) below,

$$\begin{aligned} \Phi(f) &= \sum_{e \in \mathcal{E}} (c_e(f_e) \cdot f_e + \sum_{u \in \mathcal{N}} c_e(w_u) \cdot w_u \cdot \mathbf{1}(f_u, e)) \\ &= \sum_{e \in \mathcal{E}} (c_e(f_e) \cdot f_e + \sum_{u \in \mathcal{N}} c_e(f_{e|u}) \cdot f_{e|u}), \end{aligned} \tag{3.4}$$

for each pure profile $f = (f_u)_{u \in \mathcal{N}}$. Here, we note that $f_{e|u} = w_u \cdot \mathbf{1}(f_u, e)$ for each resource $e \in \mathcal{E}$ and each player $u \in \mathcal{N}$.

We extend this potential function (3.4) of pure strategies to that of mixed profile $\pi$ as below,

$$\Phi(\pi) := \mathbf{E}[\Phi(\mathbf{f})] = \sum_{e \in \mathcal{E}} [\mathbf{E}(c_e(\mathbf{f}_e) \cdot \mathbf{f}_e) + \sum_{u \in \mathcal{N}} \mathbf{E}(c_e(\mathbf{f}_{e|u}) \cdot \mathbf{f}_{e|u})], \tag{3.5}$$

where $\mathbf{f}$ is random profile induced by mixed profile $\pi$.

Equation (3.5) actually defines a potential function on the mixed profile space of weighted congestion games satisfying Condition 1, since $\Phi(\cdot)$ is a potential function for pure profiles.

**Theorem 1.** *Equation* (3.5) *defines a 2-potential function for mixed profiles of symmetric weighted congestion games fulfilling Condition 1.*

## 3.2 An $\epsilon$-Best Response Dynamic on the Game

With the 2-potential function of mixed profiles, we are now ready to design an $\epsilon$-*approximate MINE* (see Definition 1 below) algorithm for symmetric weighted congestion games fulfilling Condition 1.

**Definition 1.** ($\epsilon$-approximate MINE) For an arbitrary constant $\epsilon \in (0, 1)$, a mixed profile $\pi$ of a symmetric weighted congestion game is an $\epsilon$-approximate MINE if, for each player $u$ and any deviation mixed strategy $\pi'_u$,

$$C_u(\pi'_u, \pi_{-u}) \geq (1 - \epsilon) \cdot C_u(\pi_u, \pi_{-u}). \tag{3.6}$$

Algorithm 1 below shows an $\epsilon$-*best response dynamic* of a symmetric weighted congestion game fulfilling Conditions 1 for an arbitrary constant $\epsilon \in (0, 1)$. It starts with an arbitrary initial mixed profile $\pi^{(0)} = (\pi_u^{(0)})_{u \in \mathcal{N}}$, and then evolves the mixed profile by iterating the following four steps over the time horizon $t \in \mathbb{N}$ until an $\epsilon$-approximate MINE of the game is met. Here, to ensure our algorithm converges within finite iterations, we impose an additional assumption on the initial mixed profile below.

**Assumption 1.** *The initial mixed profile* $\pi^{(0)}$ *fulfills the condition that* $\pi^{(0)} = (\pi_u^{(0)})_{u \in \mathcal{N}} = (\pi_{u,s_u}^{(0)})_{u \in \mathcal{N}, s_u \in \Sigma_u}$ *is a multiple of* $\delta$. *Here,* $\delta$ *is an arbitrary constant such that* $0 < \delta \ll 1$.

Note that the constant $1/\delta$ must be an integer, as $\sum_{s_u \in \Sigma_u} \pi_{u,s_u}^{(0)} = 1$ for all $u \in \mathcal{N}$ and each component of the initial profile is a multiple of $\delta$.

Here, we note that the strategy $\underline{s_u}$ can be computed efficiently by a polynomial time shortest path algorithm, e.g., Dijkstra's algorithm in [23]. Moreover, Algorithm 2 terminates within at most $1/\delta$ iterations for each player $u \in \mathcal{N}_t$. Hence, the runtime complexity of Algorithm 1 depends essentially on the total number of iterations of Algorithm 1.

**Algorithm 1.** An $\epsilon$-best response dynamic of symmetric weighted congestion game

```
Input: A symmetric weighted congestion game $\Gamma$ and two constants $\epsilon, \delta$ with $0 < \epsilon, \delta \ll 1$
Output: An $\epsilon$-approximate MINE
 1: choose a feasible mixed profile $\pi^{(0)} = (\pi_u^{(0)})_{u \in \mathcal{N}} = (\pi_{u,s_u}^{(0)})_{u \in \mathcal{N}, s_u \in \Sigma_u}$ s.t. $\pi_{u,s_u}^{(0)}$ is a multiple of $\delta$, and put $t = 0$
 2: for each $u \in \mathcal{N}$ do
 3:     compute $\underline{s_u} = \arg\min_{s_u \in \Sigma_u} c_{u,s_u}(s_u, \pi_{-u}^{(t)})$
 4: end for
 5: $\mathcal{N}_t^* = \{u \in \mathcal{N} : C_u(\underline{s_u}, \pi_{-u}^{(t)}) < (1-\epsilon) \cdot C_u(\pi^{(t)})\}$
 6: if $\mathcal{N}_t^* \neq \emptyset$ then
 7:     for each $u \in \mathcal{N}_t^*$ do
 8:         run Algorithm 2, output $\pi_u^*$
 9:     end for
10:     pick an arbitrary player $u_t$ from $\mathcal{N}_t^*$ fulfilling condition that
            $$C_{u_t}(\pi^{(t)}) - C_{u_t}(\pi_{u_t}^*, \pi_{-u_t}^{(t)}) \geq C_u(\pi^{(t)}) - C_u(\pi_u^*, \pi_{-u}^{(t)}) \quad \forall u \in \mathcal{N}_t^* \qquad (3.7)$$
11:     $\pi^{(t+1)} = (\pi_{u_t}^*, \pi_{-u_t}^{(t)})$ and $t = t + 1$
12:     return to step 2
13: else
14:     return $\pi^{(t)}$
15: end if
```

Define $T_\epsilon(\pi^{(0)}) := \arg\min\{t \in \mathbb{N} : \pi^{(t)}$ is an $\epsilon$-approximate MINE of $\Gamma\}$ and define $T_\epsilon := \max_{\pi^{(0)} \in \mathcal{F}} T_\epsilon(\pi^{(0)})$. Then $T_\epsilon(\pi^{(0)})$ is the *runtimes* of Algorithm 1 w.r.t. initial mixed profile $\pi^{(0)}$ (i.e., the number of iterations Algorithm 1 takes for finding an $\epsilon$-approximate MINE of $\Gamma$ when the initial mixed profile is $\pi^{(0)}$), and $T_\epsilon$ is the corresponding maximum runtime. Section 3.3 below inspects the upper bound of $T_\epsilon$ with the potential function $\Phi(\cdot)$ defined in equation (3.5).

## 3.3 Runtime Analysis of Algorithm 1

We first show that Algorithm 2 terminates within $\frac{1}{\delta}$ iterations, and does not fall into an infinite loop. Moreover, it indeed outputs a mixed strategy $\pi_u^*$ fulfilling the condition that $C_u(\pi_u^*, \pi_{-u}^{(t)}) < (1-\epsilon) \cdot C_u(\pi^{(t)})$. Here, $u$ is an arbitrary player in $\mathcal{N}_t^*$.

**Lemma 1.** *Consider a mixed profile $\pi^{(t)}$ with $\mathcal{N}_t^* \neq \emptyset$ and consider a player $u \in \mathcal{N}_t^*$. Let $\epsilon$ and $\delta$ be two arbitrary small constants, where $0 < \epsilon, \delta \ll 1$ and $\delta$ satisfies Assumption 1. Then, Algorithm 2 terminates within $\frac{1}{\delta}$ iterations and outputs a mixed strategy $\pi_u^*$ of the player $u$ such that $C_u(\pi_u^*, \pi_{-u}^{(t)}) < (1-\epsilon) \cdot C_u(\pi^{(t)})$.*

We then bound the potential function $\Phi(\pi)$ of an arbitrary mixed profile $\pi$ from about by $2 \cdot \text{cost}(\pi)$. While this is trivial, it will be very helpful when we derive the detailed runtime of Algorithm 1.

**Lemma 2.** *Consider an arbitrary symmetric weighted congestion game* $\Gamma$ *fulfilling Conditions 1. Let* $\Phi(\cdot)$ *be the potential function of* $\Gamma$ *as given in equation (3.5). Then, for every mixed profile* $\pi$, $1 \leq \Phi(\pi) \leq 2 \cdot cost(\pi) = 2 \cdot \sum_{u\in\mathcal{N}} C_u(\pi)$.

This naturally leads to the Lemma 3 below.

**Algorithm 2.** Local improvement

**Input:** A mixed profile $\pi^{(t)}$ with $\mathcal{N}_t^* \neq \emptyset$, a player $u \in \mathcal{N}_t^*$ and two constants $\epsilon, \delta$ with $0 < \epsilon, \delta \ll 1$.

**Output:** A mixed strategy subprofile $\pi_u^*$ of the player $u$ such that $C_u(\pi_u^*, \pi_{-u}^{(t)}) < (1-\epsilon) \cdot C_u(\pi^{(t)})$.

```
1: Initially, let $\lambda = 0$, $\pi_u^{(t,0)} = \pi_u^{(t)}$
2: for each $s_u \in \Sigma_u$ do
3:     pick an arbitrary $\overline{s_u} = \text{argmax}_{s'_u \in \Sigma_u \text{ and } \pi^{(t,\lambda)}_{u,s'_u} > 0} c_{u,s'_u}(\pi^{(t,\lambda)})$, and put
       $$\pi_{u,s_u}^{(t,\lambda+1)} := \begin{cases} \pi_{u,s_u}^{(t,\lambda)} - \delta, & \text{if } s_u = \overline{s_u}, \\ \pi_{u,s_u}^{(t,\lambda)} + \delta, & \text{if } s_u = \underline{s_u}, \\ \pi_{u,s_u}^{(t,\lambda)}, & \text{otherwise.} \end{cases}$$
4: end for
5: if $C_u(\pi_u^{(t,\lambda+1)}, \pi_{-u}^{(t)}) < (1-\epsilon) \cdot C_u(\pi^{(t)})$, then
6:     $\pi_u^* = \pi_u^{(t,\lambda+1)}$
7:     break
8: else
9:     $\lambda = \lambda + 1$ and go back to Step 2
10: end if
```

**Lemma 3.** *In every outcome* $\pi^{(t)}$, *there is a player* $u$ *with* $C_u(\pi^{(t)}) \geq \frac{\Phi(\pi^{(t)})}{2 \cdot N}$.

*Proof.* Note that $\text{cost}(\pi^{(t)}) = \sum_{u\in\mathcal{N}} C_u(\pi^{(t)})$. This together with Lemma 2 yield that

$$C_{m_t}(\pi^{(t)}) = \max_{u\in\mathcal{N}} C_u(\pi^{(t)}) \geq \frac{\text{cost}(\pi^{(t)})}{N} \geq \frac{\Phi(\pi^{(t)})}{2 \cdot N}.$$

Here, $m_t \in \mathcal{N}$ is a player with a maximum cost $C_{m_t}(\pi^{(t)}) = \max_{u\in\mathcal{N}} C_u(\pi^{(t)})$.

When $C_{u_t}(\pi^{(t)}) = C_{m_t}(\pi^{(t)})$ in each iteration $t < T_\epsilon(\pi^{(0)})$, we have

$$\Phi(\pi^{(t)}) - \Phi(\pi^{(t+1)}) = 2 \cdot (C_{m_t}(\pi^{(t)}) - C_{m_t}(\pi^{(t+1)})) \geq 2 \cdot \epsilon \cdot C_{m_t}(\pi^{(t)}) \geq \frac{\epsilon \cdot \Phi(\pi^{(t)})}{N},$$

and the potential function value decreases at a constant ratio of at least $\frac{\epsilon}{N}$ in this case. This would yield a tighter upper bound $O(\frac{N^2}{\epsilon \cdot \delta} \cdot \log(N \cdot C_{\max}))$ of $T_\epsilon$ by a similar proof to that of Theorem 2 below, where $C_{\max} = \max_{\pi \in \mathcal{F}} \max_{u \in \mathcal{N}} C_u(\pi)$ is the player's maximum expected (latency) cost value. However, in general, the selected player $u_t$ may have a (latency) cost $C_{u_t}(\pi^{(t)}) < C_{m_t}(\pi^{(t)})$. Then the above analysis does not apply. Nevertheless, Lemma 4 below shows a similar result that

$$\Phi(\pi^{(t)}) - \Phi(\pi^{(t+1)}) > \frac{\epsilon}{2 \cdot N \cdot W^2} \cdot \Phi(\pi^{(t)}) \tag{3.8}$$

for each $t < T_\epsilon(\pi^{(0)})$.

**Lemma 4.** *Consider an arbitrary symmetric weighted congestion game* $\Gamma$ *fulfilling Condition 1. We have*

$$C_{u_t}(\pi^{(t)}) - C_{u_t}(\pi^{(t+1)}) > \frac{\epsilon}{2 \cdot W^2} \cdot C_u(\pi^{(t)}) \tag{3.9}$$

*for every player* $u \in \mathcal{N}$ *and for each* $t < T_\epsilon(\pi^{(0)})$*, where* $W$ *is the common upper bound of players' weights. Moreover, Lemma 3 and equality (3.9) together yield*

$$\Phi(\pi^{(t)}) - \Phi(\pi^{(t+1)}) > \frac{\epsilon}{2 \cdot N \cdot W^2} \cdot \Phi(\pi^{(t)})$$

*for each* $t < T_\epsilon(\pi^{(0)})$ *and for each initial mixed profile* $\pi^{(0)}$*.*

Lemma 4 implies that Algorithm 1 computes an $\epsilon$-approximate MINE of the game $\Gamma$ within $O(\frac{N^2 \cdot W^2}{\epsilon \cdot \delta} \cdot \log(N \cdot C_{\max}))$ iterations when $\Sigma_u = \Sigma_v$ for all $u, v \in \mathcal{N}$. We summarize this result in Theorem 2 below.

**Theorem 2.** *Consider a symmetric weighted congestion game* $\Gamma$ *fulfilling Condition 1. Algorithm 1 computes an* $\epsilon$*-approximate MINE of* $\Gamma$ *within* $O(\frac{N^2 \cdot W^2}{\epsilon \cdot \delta} \cdot \log(N \cdot C_{\max}))$ *iterations, where* $N = |\mathcal{N}|$ *is the number of players,* $\epsilon, \delta \in (0, 1)$ *are small constants with* $\delta$ *satisfying Assumption 1, and* $C_{\max} = \max_{\pi \in \mathcal{F}} \max_{u \in \mathcal{N}} C_u(\pi)$ *is the player's maximum expected (latency) cost value.*

Theorem 2 and Lemma 4 together imply that Algorithm 1 produces an $\epsilon$-approximate MINE of $\Gamma$ within $O(\frac{N^2 \cdot W^2}{\epsilon \cdot \delta} \cdot \log(N \cdot C_{\max}))$ iterations in this particular case. We can see that this runtime is polynomial in input size when it is parameterized by $W$. Here, we note that $\epsilon$ and $\delta$ are arbitrary small constants.

## 4 Summary

We focus on the computation of approximate mixed Nash equilibria in symmetric weighted congestion games with affine linear latency functions. We design an algorithm based on best response dynamics. This algorithm is similar to that of [18]. We prove that this algorithm computes $\epsilon$-approximate mixed Nash equilibria in a polynomial runtime parameterized by the constant $W$.

To the best of our knowledge, this gives the first result for computing $\epsilon$-approximate mixed Nash equilibria for symmetric weighted congestion games, albeit under the constraints of affine linear latency functions. Nevertheless, symmetry and affine linear latency functions are still too restrictive. Thus, an interesting follow-up work is to consider weighted congestion games with more general latency functions.

**Acknowledgement.** The first author acknowledges support from the Project funded by China Postdoctoral Science Foundation No. 2024M751511. The second author acknowledges support from the National Natural Science Foundation of China with grants No. 72271085 and 61906062, and support from the Natural Science Foundation of Anhui Education Department for Excellent Young Scientisits with grant No. 2022AH020095. The third author acknowledges support from the National Natural Science Foundation of China with grant No. 72192800. The fourth author acknowledges the support from the Natural Sciences and Engineering Research Council of Canada Discovery Grant (RGPIN-2019-07115).

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Rosenthal, R.W.: A class of games possessing pure-strategy nash equilibria. Internat. J. Game Theory **2**(1), 65–67 (1973)
2. Dafermos, S.C., Sparrow, F.T.: The traffic assignment problem for a general network. J. Res. U.S. Natl. Bureau Stand. **73**(2), 91–118 (1969)
3. Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V.: Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
4. Song, X., Jiang, W., Liu, X., Lu, H., Tian, Z., Du, X.: A survey of game theory as applied to social networks. Tsinghua Sci. Technol. **25**(6), 734–742 (2020)
5. Roughgarden, T., Tardos, É.: How bad is selfish routing? J. ACM **49**(2), 236–259 (2002)
6. Correa, J.R., Schulz, A.S., Moses, N.: Selfish routing in capacitated networks. Math. Oper. Res. **29**(4), 961–976 (2004)
7. Colini-Baldeschi, R., Cominetti, R., Mertikopoulos, P., Scarsini, M.: When is selfish routing bad? the price of anarchy in light and heavy traffic. Oper. Res. **68**(2), 411–434 (2020)
8. Wu, Z., Möhring, R., Chen, Y., Xu, D.: Selfishness need not be bad. Oper. Res. **69**(2), 410–435 (2021)
9. Wu, Z., Möhring, R.: A sensitivity analysis of the price of anarchy in non-atomic congestion games. Math. Oper. Res. **48**(3), 1364–1392 (2023)
10. Wu, Z., Möhring, R., Ren, C., Xu, D.: A convergence analysis of the price of anarchy in atomic congestion games. Math. Program. **199**(1), 937–993 (2023)
11. Lu, H., Shi, Y.: Complexity of public transport networks. Tsinghua Sci. Technol. **12**(2), 204–213 (2007)
12. Nash, J.F.: Equilibrium points in $n$-person games. In: Proceedings of the National Academy of Sciences, vol. 36, pp. 48–49 (1950)
13. Chen, X., Deng, X., Teng, S.H.: Settling the complexity of computing two-player nash equilibria. J. ACM (JACM) **56**(3), 1–57 (2009)

14. Papadimitriou, C.H.: On inefficient proofs of existence and complexity classes. In: Proceedings of the 4th Czechoslovakian Symposium on Combinatorics (1991)
15. Fotakis, D., Kontogiannis, S., Spirakis, P.G.: Selfish unsplittable flows. Theoret. Comput. Sci. **348**(2–3), 226–239 (2005)
16. Fabrikant, A., Papadimitriou, C.H., Talwar, K.: The complexity of pure nash equilibria. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), pp. 604–612. ACM (2004)
17. Dunkel, J., Schulz, A.S.: On the complexity of pure-strategy nash equilibria in congestion and local effect games. Math. Oper. Res. **33**(4), 851–868 (2008)
18. Chien, S., Sinclair, A.: Convergence to approximate nash equilibria in congestion games. Games Econom. Behav. **71**(2), 315–327 (2011)
19. Caragiannis, I., Fanelli, A., Gravi, N., Skopalik, A.: Efficient computation of approximate pure nash equilibria in congestion games. In: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 532–541 (2011)
20. Caragiannis, I., Fanelli, A., Gravin, N., Skopalik, A.: Approximate pure nash equilibria in weighted congestion games: existence, efficient computation, and structure. ACM Trans. Econ. Comput. (TEAC) **3**(1), 1–32 (2015)
21. Feldotto, M., Gairing, M., Kotsialou, G., Skopalik, A.: Computing approximate pure nash equilibria in shapley value weighted congestion games. In: Proceedings of the 13th International Conference on Web and Internet Economics (WINE), pp. 191–204 (2017)
22. Ren, C., Wu, Z., Xu, D., Yang, X.: $\frac{\rho}{1-\epsilon}$-approximate pure nash equilibria algorithms for weighted congestion games and their runtimes, arXiv preprint arXiv:2208.11309
23. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)

# Dynamic Algorithms for Non-monotone Submodular Maximization

Yuanyang Liu and Wenguo Yang(✉)

School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China
liuyuanyang22@mails.ucas.ac.cn, yangwg@ucas.ac.cn

**Abstract.** Submodular maximization is a classical problem with many applications of machine learning, combinatorial optimization and so on. In recent years, there is an increasing concern on the submodular maximization problem in the dynamic setting. After the first dynamic algorithms for the submodular maximization problem was developed in 2020, Chen and Peng [3] raised an open question in 2022, asking for the possibility to extend some results from dynamic monotone submodular maximization to non-monotone cases. In this paper, we consider the problems of dynamic non-monotone non-negative submodular maximization under the cardinality and matroid constraints. We answer the open question by developing the first algorithm for non-monotone submodular maximization under the matroid constraint. We derived a randomized algorithm maintaining an $(1-\epsilon)/(8+e)$-approximate of the solution. The algorithm requires $O(k^2\epsilon^{-1}\log(k)\log^3(k/\epsilon))$ amortized oracle queries. As a byproduct, we also improved the algorithm for non-monotone submodular maximization under the cardinality constraint, maintaining an approximation guarantee of $1/(6+\epsilon)$ and requiring $O(k^2\epsilon^{-1}\log^2(k))$ amortized oracle queries which was originally developed by K. Banihashem *et al.* [5].

**Keywords:** Dynamic algorithm · Non-monotone function · Submodular maximization

## 1 Introduction

*Submodularity* is a significant property of a set function with various applications in the area of combinatorial optimization where the rank of matroids, edge cuts, coverage and so on are all instances of submodular functions. It is also widely used in the area of data summarization, information gathering and social networks. A function is called *submodular* if for all $A \subseteq B \subseteq V$ and $e \notin B$, $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$, which intuitively implies that the same element not in the current set will bring less marginal value as the current set containing more elements.

Given a submodular function $f$ defined over a ground set $V$, in the *submodular maximization problem*, we need to compute an optimal solution $S \subseteq V$, which maximizes the value $f(S)$.

The celebrated work given by Fisher *et al.* [9] in the 1970s, was the first study considering the submodular maximization problem in the offline model. They analyzed a greedy heuristic for the problem and obtained an approximation ratio of $1 - 1/e$ as a result.

However, because of the rapid development of the computer science technology, the algorithm has to be able to deal with massive datasets which vary constantly. As a result, recent researches have developed several big data models to solve the problem. These include streaming models, dynamic models, parallel models, online models and so on.

In the fully dynamic model, there is a sequence of updates of the ground set $V$ denoted by $\varXi$. Each update causes an insertion or deletion of an element $e$ to the previous ground set $V$. The goal of the algorithms is to update the optimal solution after each insertion or deletion, and maintain the approximation guarantee in the entire process. The update time is measured by the amortized query complexity, which counts the average number of queries required by the algorithm during each update.

### 1.1 Related Work

The study of the dynamic submodular maximization was initiated in 2020 based on two works by Lattanzi *et al.* [10] and Monemizadeh [4]. They both provided an algorithm for the dynamic monotone submodular maximization problem under the cardinality constraint $k$ with an approximation of $1/(2+\epsilon)$. Their amortized query complexity are respectively $O(\epsilon^{-11}\log^6(k)\log^2(n))$ and $O(k^2\epsilon^{-3}\log^5(n))$. In 2023 Banihashem *et al.* [1] developed an algorithm for the same problem, improving the amortized query complexity to $O(k\epsilon^{-1}\log^2(k))$, which is the first one to be independent of $n$ (the size of the ground set $V$).

Considering the problem of dynamic monotone submodular maximization under the matroid constraint, Banihashem *et al.* [1] and Dütting *et al.* [6] both developed algorithms with an approximation guarantee of $1/(4+\epsilon)$. Their amortized query complexity are $O(k\log(k)\log^3(k/\epsilon))$ and $O(k^2\epsilon^{-1}\log(k)\log^2(n)\log^3(k/\epsilon))$, respectively.

For non-monotone cases, Banihashem*et al.* [5] demonstrated a reduction from the monotone problem under the cardinality constraint to the non-monotone problem under the same constraint. By exploiting the algorithm in [1], they obtained an approximation guarantee of $1/(8+\epsilon)$ with an amortized query complexity of $O(k^2\epsilon^{-2}\log^3(k))$ per update.

### 1.2 Our Contribution

In this paper, we consider the dynamic non-monotone submodular maximization problem under cardinality and matroid constraints. In the model, we have a *universal ground set* $\mathcal{V}$. At the beginning of the algorithm, the *ground set* is denoted by $V$. The ground set is going to have a sequence of updates in the dynamic model, each of which cause an insertion or deletion of a single element.

At time $t$, we denote $V_t$ as the current ground set which contains the elements that are inserted but not deleted from the set $V$ till time $t$.

We assume that there is a non-monotone non-negative submodular function $f$ defined on the set $\mathcal{V}$. Then the goal of our algorithm is to compute a subset of $V_t$ under some constraints, the submodular value of which is the maximum of all those subsets under the constraint.

Calculating such a subset is known to be NP-hard even in the offline setting [8], so our focus comes to developing approximate algorithms with a fast update time, which is measured by the property of query complexity. As many existing results are for monotone submodular maximization, it becomes more challenging to deal with non-monotone cases as adding a new element to the current set may decrease the submodular value.

Inspired by the works of Banihashem *et al.* [1,5], we extend their algorithm from monotone to non-monotone cases. Using the idea of Gupta *et al.*, we create two parallel instances of the algorithm. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ denote the two instances. First, with input $V$, we run the algorithm, which is regulated from those in the monotone cases, in $\mathcal{I}_1$ to obtain a solution $S_1$. Then, we delete the elements of $S_1$ in the instance $\mathcal{I}_2$. That is, we run the algorithm in $\mathcal{I}_2$ with $V \setminus S_1$ as the input to obtain $S_2$. Intuitively, the set $S_2$ might be a better solution if $S_1 \cap S^*$ behaves bad, where $S^*$ denotes the accurate optimal solution. At this time, we can run offline algorithms on $S_1$ to obtain a subset $S_3$, which bounds the value $f(S_1 \cap S^*)$. Finally, we let $S = \text{argmax}_{C \in \{S_1, S_2, S_3\}} f(C)$ be the output of our algorithm.

In this paper, we present our algorithms for both cardinality and matroid constraints. For the cardinality case, we improved the result of [5], reaching an approximation guarantee of $1/(6 + \epsilon)$ rather than $1/(8 + \epsilon)$. For the matroid case, we developed the first algorithm for the problem using the idea of leveling algorithm, reaching an approximation guarantee of $(1 - \epsilon)/(8 + e)$.

In Sect. 2, we outline the basics of the problem of dynamic non-monotone submodular maximization under cardinality and matroid constraints. Section 3 shows the algorithms we used in our reduction. In Sect. 4, we analyze the approximation guarantee and the amortized query complexity of our algorithms.

## 2 Preliminaries

**Submodular Function.** Let $V$ be a set of elements called *ground set*. We call a function $f : 2^V \to \mathbb{R}^{\geq 0}$ *submodular* if for all $A, B \subseteq V$, $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$. Equivalently, for any subsets $A \subseteq B \subseteq V$ and any element $e \in V \setminus B$, $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. Function $f$ is *monotone* if $f(A) \leq f(B)$holds for every$A \subseteq B$, otherwise it is called *non-monotone*. In this paper, we assume that the submodular function is non-negative, which requires the value $f(S) \geq 0$, for all $S \subseteq V$.

**Matroid.** A *matroid* $\mathcal{M}(V,\mathcal{I})$ consists of a ground set $V$ and a nonempty downward-closed set system $\mathcal{I} \subseteq 2^V$ (which means $A \in \mathcal{I}$ can be inferred from

$B \in \mathcal{I}$ if $A \subseteq B \subseteq V$) known as the *independent sets*. The system is required to satisfy the *exchange axiom*: if $A, B$ are independent sets and $|A| < |B|$, there exists an element $x \in B \setminus A$ such that $A \cup \{x\}$ is an independent set. A subset of $V$ is called *dependent* if it is not an independent set. An independent set which becomes dependent when adding any other element in $V$ is called a *basis* for the matroid. A dependent set which becomes independent when deleting any element inside is called a *circuit* for the matroid. Let $A$ be a subset of $V$, then the rank of $A$, denoted by $\text{rank}(A)$, is defined as the maximum cardinality of an independent subset of $A$.

**Access Model.** In this paper, we assume that the access to a submodular function is given by an *oracle*. The oracle allows set queries such that for every subset $A \subseteq V$, one can query the value $f(A)$. Since the marginal value $\Delta_f(e|A)$ is defined as $f(A \cup \{e\}) - f(A)$, it requires two queries, $f(A \cup \{e\})$ and $f(A)$.

**Dynamic Submodular Maximization Under Cardinality and Matroid Constraints.** Let $f$ be a submodular set function. The purpose of the submodular maximization is to compute a subset $S^*$ of the ground set $V$ as the optimal solution to maximize the value $f(S)$ under some constraints. In the dynamic model, we assume that the ground set has a sequence of updates. We denote it as $\Xi = \{\Xi_1, \Xi_2, ...\}$, each of which refers to an insertion or deletion of a single element in the ground set $V$. We define $V_t$ as the set of elements of $V$ and those have been inserted but not deleted from $V$ after the $t^{th}$ update. To make it clear, we denote the universal ground set as $\mathcal{V}$, which includes all the elements involved in the sequence $\Xi$ and $V$. The purpose of the dynamic submodular maximization problem is to find the optimal solution after every update, i.e., to find a subset $S_t$ that $f(S_t) = \max_{S \in V_t} f(S)$, where $S_t$ is restricted by some constraints. For the cardinality constraint, there is an integer $k$ and $|S_t| \leq k$ is required. For the matroid constraint, there is a matroid defined on the set $\mathcal{V}$, and $|S_t|$ needs to be an independent set, that is, $S_t \in \mathcal{I}$ is required by the constraint.

**Approximation Guarantee.** An algorithm of the problem of submodular maximization is an $\alpha$-approximation algorithm, if the solution $S$ is guaranteed to satisfy $f(S) \geq \alpha \cdot OPT$. In the dynamic model, the inequality needs to hold at any time $t$. For randomized algorithms, $\alpha$-approximation algorithm is defined as those satisfying $\mathbb{E}[f(S)] \geq \alpha \cdot OPT$, where $\mathbb{E}[f(S)]$ denotes the expectation of the random variable $f(S)$.

**Query Complexity.** The *query complexity* is the number of the oracle queries required by the algorithm to compute the solution. For the matroid constraint, the oracle queries include the submodular oracle and the independent oracle. The amortized query is defined as the average number of oracle queries required for each update. This property helps to measure the time complexity of a dynamic algorithm.

## 3 Dynamic Algorithms

In this section, we introduce the *leveling algorithm* we use to deal with the problems of dynamic submodular maximization under the cardinality and matroid constraints. The algorithm is inspired by the works of [1] and [5]. However, the original algorithms were developed for the monotone cases, so we regulated some parts to apply to the non-monotone occasions.

The leveling algorithms are based on random permutations of the elements. In a random order, we take each element into consideration. The useful elements are defined as *promoting elements* which will be formally discussed later. As soon as we find a promoting element, we build a new level and continue iterating the elements until all the elements in $V$ have been considered. In the end, we will obtain a set $S$, which consists of some (or all) of the promoting elements as the result of the algorithm. After each update of the ground set, we make several regulations to our data structure, in order to maintain the approximation ratio at any time $t$.

However, in the preceding descriptions, we assume that we have known the optimal value $f(S^*) = OPT$ in advance, while in fact the value tend to be unknown for us and may change after each update. In order to relax the assumption, we prepare several parallel runs of the algorithm, with a different guess of the optimal value in each one of the runs. During an update, we only need to consider those runs in a range of guesses depending on the properties of the element $e$ involved in the update.

### 3.1 The Leveling Algorithm Under the Cardinality Constraint

In this subsection, we introduce a leveling algorithm regulated from the works of Banihashem *et al.* [1] which was developed for monotone situations. In the case of the cardinality constraint, the algorithm constructs a set of levels $L_0, L_1, ..., L_T$, where $T$ is the number of the nonempty levels. Every $L_l$ consists of two sets $R_l$, $I_l$ and an element $e_l$. They satisfy the following properties:

1. $R_0 = V, I_0 = \emptyset$
2. $R_0 \supseteq R_1 \supset ... \supset R_T \supset R_{T+1} = \emptyset$
3. For $1 \leq l \leq T$, we have $I_l = I_{l-1} \cup \{e_l\}$

In this algorithm, $I_T$ is the solution. During the construction of each level, the key concept is the notion of *promoting elements.*

**Definition 1.** *(Promoting elements) Let $\tau > 0$ be a parameter and $k$ be the cardinality constraint. For a set $I_l, 1 \leq l \leq T$, an element $e$ is a promoting element, if $f(I_l \cup \{e\}) - f(I_l) \geq \tau$ and $|I_l| < k$.*

Here, $\tau$ is a threshold parameter to guarantee the lower bound of the submodular value of the function. Now the levels can be constructed as follows: Let $l = 1$, and $R_1 = \{e \in R_0 : f(e) > \tau\}$. Then, we permute the elements of $R_1$ in a random order and let the first element be $e_1$ and the set $\{e_1\}$ be $I_1$. We iterate through the elements for the permutation and for every element $e \in R_1$, we check whether $e$ is a promoting element of the set $I_1$.

If $e$ is a promoting element, we let $e_l$ be $e$ and $I_l$ be $I_{l-1} \cup \{e_l\}$. Then, we create the next level $L_{l+1}$ by setting $R_{l+1} = \emptyset$ and let $l$ iterate to $l+1$.

If $e$ is not a promoting element, we need to find the smallest $z$ such that $e$ is not a promoting element of $I_z$. According to submodularity, there must be a smallest $z$ such that $e$ is not a promoting element of $I_z$, and $e$ is not a promoting element of all $I_{\geq z}$. Once we find such $z$, we add $e$ to all the sets $R_r$ for $2 \leq r \leq z$.

During updates, we need to invoke the insertion and deletion subroutines. If we delete an element $v$, we first delete $v$ in the set $R_0$, and check if the element $v$ was in the previous $I_T$. In other words, to check if there exists $i, 1 \leq i \leq T$, such that $e_i = v$.

If not, it is simple that we only need to delete the element in each level. That is, to delete $v$ from those sets $R_i$ which includes the element $v$.

However, if there exists $i, 1 \leq i \leq T$ such that $e_i = v$, this means that the leveling data structure has been broken. Then we need to rebuild all the levels $L_{i \leq j \leq T}$. We do this by invoking the level construction function of the level $L_i$, which permutes $R_i$ randomly and calculates new $I_l, R_l, e_l$ and so on to reconstruct the levels.

For the case of insertion, the update works similarly. When we insert an element $v$, we need to check each level $L_j$ if the element $v$ is a promoting element of the set $I_j$. If so, we add $v$ into the set $R_{j+1}$ with probability $1/|R_i|$.

## 3.2 Non-monotone Cases Under the Cardinality Constraint

In the non-monotone cases, letting $\tau = OPT/((4 + 1/\alpha)k)$, we create two independent instances $\mathcal{I}_1$ and $\mathcal{I}_2$. The solutions given by the algorithm in each instance are denoted by $S_1$ and $S_2$ respectively.

At the beginning of the algorithm, we run the algorithm in $\mathcal{I}_1$ with the ground set $V$ as the input. The algorithm returns a set $S_1$ as the solution. Then, we run the algorithm in $\mathcal{I}_2$ with the ground set $V \setminus S_1$ as the input, obtaining a set $S_2$ which satisfies $S_1 \cap S_2 = \emptyset$. Next, we run unconstrained offline algorithms on $S_1$ to obtain a set $S_3 \subseteq S_1$. Finally, we let $S = \text{argmax}_{C \in \{S_1, S_2, S_3\}} f(C)$ be the solution.

Consider the subroutines of insertions deletions. Let $\textsc{Insert}_{\mathcal{I}_i}(v)$ and $\textsc{Delete}_{\mathcal{I}_i}(v)$ denote the subroutines in the preceding subsection. Let $\textsc{Extract}_{\mathcal{I}_i}$ denote the process of returning an optimal solution $S_i$ in the instance $\mathcal{I}_i$. When we insert an element $v$, we first run the insertion subroutine in $\mathcal{I}_1$ to obtain an updated $S_1$. Let $S_1^-$ denote the previous $S_1$ before the update. To update $\mathcal{I}_2$, we first insert (or delete) $v$ in (the ground set of) $\mathcal{I}_2$, then insert the elements of $S_1^- \setminus S_1$ one by one into $\mathcal{I}_2$. Next, we delete the elements of $S_1 \setminus S_1^-$ in $\mathcal{I}_2$. Note that $S_1 \cap S_2 = \emptyset$ holds after every update, which is useful in our analysis. After this, we still run unconstrained offline algorithms on $S_1$ to obtain a set $S_3 \subseteq S_1$ and let $S = \text{argmax}_{C \in \{S_1, S_2, S_3\}} f(C)$ be the solution.

For the offline algorithm, we can use the method developed by Buchbinder *et al.* [7]. In this algorithm, we start with two sets, $\emptyset$ and $S_1$, and consider the elements of $S_1$ one at a time. For each element, we decide to delete it from the second set or insert it to the first set randomly, with the probability associated with the submodular value of the set. Then, we can obtain a 1/2-approximate solution, which means letting $\alpha = 1/2$. Another method chooses every element randomly with a probability of 1/2, obtaining a 1/4-approximate solution, which means letting $\alpha = 1/4$.

In our assumption, the parameter $\tau$ depends on the value $OPT$. But in reality, the value $OPT$ is unknown in most of the situations. Using the technique in [10], we can run parallel instances of our algorithm. First we make some parallel guesses of the value $OPT$. Let the guesses be $(1 + \epsilon)^i$ where $i \in \mathbb{Z}$. For an element $e$, we can ignore the guesses in which the $OPT$ satisfies either $f(e) < \tau$ (where $e$ will never be a promoting element) or $f(e) > OPT$ (where the guess is conflicted with $e$).

## 3.3 The Leveling Algorithm Under the Matroid Constraint

The problems under the matroid constraint are similar to those under the cardinality constraint. However, it is more complicated. Unlike the occasions under the cardinality constraint, we are not able to add elements freely into the current set as it may break the independence of the set.

We also introduce a leveling algorithm under the matroid constraint. The algorithm constructs a set of levels $L_0, L_1, ..., L_T$, where $T$ is the number of the nonempty levels. Every $L_l$ consists of sets $R_l$, $I_l$, $I'_l$ and an element $e_l$. They satisfy the following properties:

1. $R_0 = V, I_0 = \emptyset$
2. $R_0 \supseteq R_1 \supset ... \supset R_T \supset R_{T+1} = \emptyset$
3. For $1 \leq l \leq T$, we have $I_l = I_{l-1} \cup \{e_l\}$
4. $I_i \in \mathcal{I}, I'_i = \cup_{j \leq i} I_j$

In this algorithm, $I_T$ is the solution. The key concept in the matroid case is still *promoting elements*. Compared to the cardinality case, the concept is more complicated, since an element $e$ cannot be added into the current set freely. Before the definition, we define the *weight* of an element $e$ in $I'_T$, which is denoted as $w(e)$. We let $w(e) = f(I'_l + e) - f(I'_l)$ when $e$ is added into $I'_l$.

**Definition 2.** *(Promoting elements) For a level $L_l, 1 \leq l \leq T$, an element $e$ is a promoting element for the level, if property 1 and either 2 or 3 hold.*

*Property 1: $f(I'_l + e) - f(I'_l) \geq \tau$.*

*Property 2: $I_l + e$ is independent.*

*Property 3: $I_l + e$ is not independent, and there exists an element $\hat{e}$, such that $I_l + e - \hat{e}$ is independent and $w(\hat{e}) \leq 1/2 \cdot (f(I'_l + e) - f(I'_l))$.*

The idea of the algorithm was affected by the *swap algorithm*, which allows an element in the current solution to swap the new element to keep the property of independent set. The level construction and update algorithms are similar to that of the cardinality case, refer to [1] for more details.

### 3.4 Non-monotone Cases Under the Matroid Constraint

In this section, we extend the algorithm in the last subsection to non-monotone cases.

Different from the cardinality cases, the sets $I_T$ and $I'_T$ are both significant to the analysis for the algorithm, so the situation becomes more complicated when it comes to the matroid constraint. Still, we create two independent instances, $\mathcal{I}_1$ and $\mathcal{I}_2$. However, the set $I_T$, $I'_T$ given by the algorithm in each instance are denoted by $S_1$, $S'_1$ and $S_2$, $S'_2$ respectively. Note that $S_1$ and $S_2$ are independent sets while $S'_1$ and $S'_2$ are not necessarily independent.

As we did to deal with cardinality cases, we run the algorithm in $\mathcal{I}_1$ with the ground set $V$ as the input. The algorithm returns $S_1$ and $S'_1$. Then, we run the algorithm in $\mathcal{I}_2$ with the ground set $V \setminus S'_1$ as the input, obtaining $S_2$ and $S'_2$ which satisfies $S'_1 \cap S'_2 = \emptyset$. Next, we run offline algorithms on $S'_1$ to obtain a set $S_3 \subseteq S'_1$. Finally, we let $S = \text{argmax}_{C \in \{S_1, S_2, S_3\}} f(C)$ be the solution.

The update process is the same as the cardinality case, so we skip the introduction. However, the offline on the set $S'_1$ is not as simple as the cardinality case since the set $S'_1$ is probably not an independent set. Fortunately, the size can be upper-bounded by the rank of the matroid $k$ instead of the size of the ground set $n$. So we use the offline continuous greedy algorithm developed by Feldman *et al.* with an approximation guarantee of $1/e$.

## 4 Analysis of the Algorithms

### 4.1 Approximation Guarantee

In this subsection, we prove the approximation guarantees of our algorithms.

**Cardinality Case.** We first consider the solution given by the leveling algorithm. In fact, the algorithm stops only when there is no *promoting elements* available in the candidates. Recall the definition of the promoting elements, one of these properties must hold:

1. $|I_T| = k$.
2. For all $e \in R_T \subset \{e_T\}$, $f(I_T \cup \{e\}) - f(I_T) < \tau$.

So we can obtain the following theorem:

**Theorem 1.** *The solution given by the cardinality leveling algorithm satisfies exactly one of the two properties:*

1. $|I_T| = k$ *and* $f(I_T) \geq \tau k$.
2. $|I_T| < k$ *and for all* $e \in V \setminus I_T$, $f(I_T \cup \{e\}) - f(I_T) < \tau$.

*Proof.* As the preceding analysis, we know that the algorithm only stops when there are no promoting elements. In the first case, $|I_T| = k$. According to the algorithm, we have $f(I_l) - f(I_{l-1}) \geq \tau$ for $T = 2, ..., T$ and $f(I_1) \geq \tau$. After adding all the equations together, we obtain that $f(I_T) \geq \tau k$.

In the second case where $|I_T| < k$, for all $e \in R_T \setminus \{e_T\}$, $f(I_T \cup \{e\}) - f(I_T) < \tau$. According to the algorithm, $f(I_T \cup \{e\}) - f(I_T) \geq \tau$ means that $e$ is a promoting element of $I_T$, so it must be included in the set $R_T$. As there is no such element in $R_T$, we prove that the property hold for all the elements in $V$.

Using this property, we obtain the following result:

**Theorem 2.** *Suppose the optimal value* $OPT^* \in [\frac{OPT}{1+\epsilon}, OPT]$. *When applying the cardinality algorithm to the non-monotone cases, letting* $\tau = OPT/((4 + 1/\alpha)k)$*, the expected submodular value of the solution is* $\mathbb{E}[f(S)] \geq (1-O(\epsilon))/(4+1/\alpha) \cdot OPT^*$. $\alpha = 1/2$ *or* $1/4$ *depending on the offline algorithm we choose.*

*Proof.* Using Jensen's inequality, we have

$$\mathbb{E}[\max(f(S_1), f(S_2), f(S_3))] \geq \max(\mathbb{E}[f(S_1)], \mathbb{E}[f(S_2)], \mathbb{E}[f(S_3)]) \tag{1}$$

Thus, we just need to prove

$$\max(\mathbb{E}[f(S_1)], \mathbb{E}[f(S_2)], \mathbb{E}[f(S_3)]) \geq (1 - O(\epsilon))/(4 + 1/\alpha) \cdot OPT \tag{2}$$

Then, let $S^*$ denote the optimal solution. We consider two cases: (i) $f(S_1 \cap S^*) \geq \tau k/\alpha$ (ii) $f(S_1 \cap S^*) < \tau k/\alpha$.

If (i) is true, we have

$$\begin{aligned}\mathbb{E}[f(S_3)] &\geq \alpha \max_{C \in S_1}(f(C)) \\ &\geq \alpha f(S_1 \cap S^*) \\ &\geq \alpha \tau k/\alpha \\ &= \tau k = 1/(4 + 1/\alpha) \cdot OPT \\ &\geq 1/(4 + 1/\alpha) \cdot OPT^*\end{aligned} \tag{3}$$

If (ii) is true, we consider whether one of the sets $S_1$ and $S_2$ has the cardinality $k$. Let $|S_i| = k$, we have

$$f(S_i) \geq \tau k = 1/(4 + 1/\alpha) \cdot OPT \geq 1/(4 + 1/\alpha) \cdot OPT^* \tag{4}$$

Otherwise, we can assume that $|S_1|, |S_2| < k$. Then, according to the previous theorem, for all $e \in V \setminus S_1$, $f(S_1 \cup \{e\}) - f(S_1) < \tau$. Especially, for all $e \in S^* \setminus S_1$, $f(S_1 \cup \{e\}) - f(S_1) < \tau$. According to submodularity, we have

$$\begin{aligned}f(S_1 \cup S^*) - f(S_1) &\leq \sum_{e \in S^* \setminus S_1} (f(S_1 \cup \{e\}) - f(S_1)) \\ &\leq |S^* \setminus S_1| \cdot \tau \\ &\leq \tau k\end{aligned} \tag{5}$$

That is, $f(S_1) \geq f(S_1 \cup S^*) - \tau k$. Similarly, we have $f(S_2) \geq f(S_2 \cup (S^* \setminus S_1)) - \tau k$.

Thus,

$$\begin{aligned} f(S_1) + f(S_2) &\geq f(S_1 \cup S^*) + f(S_2 \cup (S^* \setminus S_1)) - 2\tau k \\ &\geq f(S_1 \cup S^* \cup S_2) + f(S^* \setminus S_1) - 2\tau k \\ &\geq f(S^* \setminus S_1) - 2\tau k \end{aligned} \tag{6}$$

According to submodularity,

$$f(S^* \setminus S_1) + f(S_1 \cap S^*) \geq f(S^*) = OPT^* \tag{7}$$

So we have

$$\begin{aligned} f(S_1) + f(S_2) &\geq f(S^* \setminus S_1) - 2\tau k \\ &\geq OPT^* - f(S_1 \cap S^*) - 2\tau k \\ &> OPT^* - \tau k/\alpha - 2\tau k \\ &\geq (1 - (1+\epsilon)(1/(4+1/\alpha)/\alpha + 2/(4+1/\alpha))) \cdot OPT^* \\ &\geq (2 - \epsilon(2+1/\alpha))/(4+1/\alpha) \cdot OPT^* \\ &= 2(1-O(\epsilon))/(4+1/\alpha) \cdot OPT^* \end{aligned} \tag{8}$$

Therefore,

$$\max(f(S_1), f(S_2)) \geq \frac{f(S_1) + f(S_2)}{2} \geq (1-O(\epsilon))/(4+1/\alpha) \cdot OPT^* \tag{9}$$

This leads to

$$\max(\mathbb{E}[f(S_1)], \mathbb{E}[f(S_2)], \mathbb{E}[f(S_3)]) \geq (1-O(\epsilon))/(4+1/\alpha) \cdot OPT^* \tag{10}$$

So we have

$$\mathbb{E}[f(S)] \geq (1-O(\epsilon))/(4+1/\alpha) \cdot OPT^* \tag{11}$$

Thus, if we use the local search method as the offline algorithm where $\alpha = 1/2$, we obtain a randomized $(1-O(\epsilon))/6$-approximate algorithm for the non-monotone maximization under the cardinality constraint.

**Matroid Case.** The problem is much more intricate under the matroid constraint as we cannot directly separate the solution into two kinds as we did in the cardinality constraint considering the complexity of the matroids. As a result, we need more detailed properties of the sets $I_T$ and $I'_T$ given by the leveling algorithm.

**Definition 3.** *For each element $e \in V$, we define $z(e) = \max\{i : e \in R_i\}$. If $e_{z(e)} = e$, then in the algorithm $w(e) = f(I'_{z(e)-1} + e) - f(I'_{z(e)-1})$. For other $e \in V$, define $w(e) = f(I'_{z(e)} + e) - f(I'_{z(e)})$. For a set $E \subseteq V$, define $w(E) = \sum_{e \in E} w(e)$.*

After the definition of $z(e)$ and $w(e)$, we have made the preparations to prove the following theorem:

**Theorem 3.** *Let $E \subseteq V$ be an arbitrary independent set, $S$, $S'$ be an output of the leveling algorithm, we have*

$$f(E \cup S') \leq 4f(S) + \tau k \tag{12}$$

With the boundary given by this theorem, we can easily give an approximation guarantee of our algorithm. The proof of the theorems involves three lemmas and are similar to those in [1]. The lemmas are as follows:

**Lemma 1.** *In the leveling algorithm, we have*

$$f(S') = w(S') \leq 2w(S) \leq 2f(S) \tag{13}$$

**Lemma 2.** *Let $E \in V$ be an arbitrary independent set, $S$, $S'$ be an output of the leveling algorithm, we have*

$$f(E \cup S') \leq 2w(S) + w(E) \tag{14}$$

**Lemma 3.** *Let $E \in V$ be an arbitrary independent set, $S$ be an output of the leveling algorithm, we have*

$$w(E) \leq 2w(S) + \tau k \tag{15}$$

Similar to the analysis of Theorem 2, we obtain the following result:

**Theorem 4.** *Suppose the optimal value $OPT^* \in [\frac{OPT}{1+\epsilon_1}, OPT]$. When applying the matroid algorithm to the non-monotone cases, letting $\tau = \epsilon_2/(2k) \cdot OPT$, the expected submodular value of the solution is $\mathbb{E}[f(S)] \geq (1 - O(\epsilon_1) - O(\epsilon_2))/(8 + 1/\alpha) \cdot OPT^*$.*

This means that our randomized algorithm has an approximation of $(1 - \epsilon)/(8 + 1/\alpha)$. Using the continuous greedy algorithm [2] where $\alpha = 1/e$, we obtain an $(1 - \epsilon)/(8 + e)$-approximate algorithm.

## 4.2 Query Complexity

The query complexity is the property to measure the update time of an algorithm. In this subsection, we compute the query complexity of our algorithms for the problems under cardinality and matroid constraints respectively.

**Cardinality Case.** According to the work of [1], in an update, the expected queries required by the cardinality leveling algorithm is $O(k \log(k))$. In our algorithm, an update causes one update in the instance $\mathcal{I}_1$, and causes at most $(1 + |S_1 \setminus S_1^-| + |S_1^- \setminus S_1|)$ updates in $\mathcal{I}_2$, which is bounded by $2k + 1$. Therefore, counting the queries of the two instances, we obtain an amortized query complexity of $O(k^2 \log(k))$. Since the query complexity of the offline algorithm is $O(k)$, the sum of them is still $O(k^2 \log(k))$.

Now we relax the $OPT$ assumption. As discussed, we only need to update the guesses where $f(e) \leq OPT \leq (4 + 1/\alpha)kf(e)$. Thus, the factor here is $O(\epsilon^{-1} log(k))$.

Then we obtain the result that the amortized query complexity of the algorithm is $O(k^2\epsilon^{-1} \log^2(k))$.

**Matroid Case.** First, we bound the size of $T$. As $\tau = \epsilon_2/(2k) \cdot OPT$ and $\max_{e\in V} f(e) \leq OPT$, we have

$$\frac{\max_{e\in I'_T} f(e)}{\min_{e\in I'_T} f(e)} \leq \frac{OPT}{\tau} = \frac{2k}{\epsilon_2} \tag{16}$$

During the leveling algorithm, each element $e$ in $I_l$ is replaced at most once by another element whose weight is at least $2w(e)$. So if we consider the chain formed by swapping, the length of the chain $c$ is at most $O(\log(k/\epsilon_2))$ since $2^c \leq 2k/\epsilon_2$. As the elements in $I_T$ is at most $k$, each forming a chain whose length is at most $O(\log(k/\epsilon_2))$, the number of the elements in $I'_T(T)$ is $O(k \log(k/\epsilon_2))$.

Using the analysis in [1], given a specified $OPT$, the expected number of queries is at most $O(k \log(k) \log^2(k/\epsilon_2))$. During the update of our non-monotone algorithm, for the same reason as the cardinality case, an update will cause at most $O(k)$ insertion or deletions in $\mathcal{I}_1$ and $\mathcal{I}_2$, and the amortized query complexity is $O(k^2 \log(k) \log^2(k/\epsilon_2))$.

Finally, we relax the $OPT$ assumption. We make some parallel guesses of the value $OPT$. Let the guesses be $(1 + \epsilon_1)^i$ where $i \in \mathbb{Z}$. This will add a factor $O(\epsilon_1 \log(k/\epsilon_2))$ to our complexity. Considering all the queries, the amortized query complexity is $O(k^2\epsilon_1^{-1} \log(k) \log^3(k/\epsilon_2))$.

## 5 Conclusion

In this paper, we studied the dynamic algorithms for the problem of non-monotone submodular maximization under cardinality and matroid constraints by demonstrating a reduction from the monotone cases. As a result, we obtained two algorithms, respectively for the problem of non-monotone submodular maximization under cardinality and matroid constraints, with an approximation of $1/(6 + \epsilon)$ and $(1 - \epsilon)/(8 + e)$. The amortized query complexity of the algorithms are $O(k^2\epsilon^{-1} \log^2(k))$ and $O(k^2\epsilon^{-1} \log(k) \log^3(k/\epsilon))$.

## References

1. Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., Monemizadeh, M.: Dynamic algorithms for matroid submodular maximization. In: Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024 (2024)

2. Feldman, M., Naor, J., Schwartz, R.: A unified continuous greedy algorithm for submodular maximization. In: Proceedings of 52nd Annual IEEE Symposium Foundations of Computer Science (FOCS), pp. 570–579 (2011)
3. Chen, X., Peng, B.: On the complexity of dynamic submodular maximization. In: STOC 2022: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, pp. 1685–1698 (2022)
4. Monemizadeh, M.: Dynamic submodular maximization. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems (2020)
5. Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., Monemizadeh, M.: Dynamic non-monotone submodular maximization. In: Thirty-Seventh Conference on Neural Information Processing Systems (2023)
6. Dütting, P., Fusco, F., Lattanzi, S., Norouzi-Fard, A., Zadimoghaddam, M.: Fully dynamic submodular maximization over matroids (2023)
7. Buchbinder, N., Feldman, M., Seffi, J., Schwartz, R.: A tight linear time (1/2)-approximation for unconstrained submodular maximization. SIAM J. Comput. **44**(5), 1384–1402 (2015)
8. Feige, U., Mirrokni, V.S., Vondrák, J.: Maximizing non-monotone submodular functions. SIAM J. Comput. **40**(4), 1133–1153 (2011)
9. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions - I. Math. Program. **14**(1), 265–294 (1978)
10. Lattanzi, S., Mitrovic, S., Norouzi-Fard, A., Tarnawski, J., Zadimoghaddam, M.: Fully dynamic algorithm for constrained submodular optimization. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020 (2020)

# Broadcasting and Three List Subtraction

Hovhannes A. Harutyunyan and Narek Hovhannisyan(✉)

Department of Computer Science and Software Engineering, Concordia University, Montreal, QC H3G 1M8, Canada
haruty@cs.concordia.ca, narekh98@gmail.com

**Abstract.** Broadcasting is an information dissemination primitive where a message is passed from one node (called originator) to all other nodes in the network. In the scope of this paper, we will mainly focus on determining the broadcast time and the optimal broadcasting scheme for graphs. Determination of the broadcast time of a node in an arbitrary network is known to be NP-hard. Polynomial time solutions are known only for a few classes of networks. In this paper, we will consider networks that can be represented as $k$-path graphs. We will pose a new problem, called 3 list subtraction, and discuss its relation to the broadcast time problem on $k$-path graphs.

**Keywords:** Interconnection networks · Information dissemination · Broadcasting · List subtraction

## 1 Introduction

Broadcasting is one of the most important information dissemination processes in an interconnected network. Over the last four decades, a large amount of research work has been published concerning broadcasting in networks under different models. Models can have different numbers of originators, numbers of receivers at each time unit, distances of each call, numbers of destinations, and other characteristics of the network such as the knowledge of the neighborhood available to each node. In the context of this paper, we are going to focus on the classical model of broadcasting. The network is modeled as an undirected connected graph $G = (V, E)$, where $V(G)$ and $E(G)$ denote the vertex set and the edge set of $G$, respectively. The classical model follows the below-mentioned basic assumptions.

(1) The broadcasting process is split into discrete time units.
(2) The only vertex that has the message (is *informed*) at the first time unit is called *originator*.
(3) In each time unit, an informed vertex (*sender*) can *call* at most one of its uninformed neighbors (*receiver*).
(4) During each unit, all calls are performed in parallel.
(5) The process halts as soon as all the vertices in the graph are informed.

We can represent each call in this process as an ordered pair of two vertices $(u, v)$, where $u$ is the sender and $v$ is the receiver. The *broadcast scheme* is the order of calls made by each vertex during a broadcasting process and can be represented as a sequence $(C_1, C_2, ..., C_t)$, where $C_i$ is the set of calls performed in time unit $i$. An informed vertex $v$ is *idle* in time unit $t$ if $v$ does not make any call in time $t$. A broadcast scheme is called *busy* if any informed vertex sends a message to one of its uninformed neighbors during each round. These schedules guarantee that as long as there remains an uninformed neighbor, vertices are never idle.

Any broadcast scheme forms a directed spanning tree (*broadcast tree*) rooted at the originator. We are also free to omit the direction of each call in the broadcast tree.

The *broadcast time* of a vertex $v$ in a given graph $G$ is the minimum number of time units required to broadcast in $G$ if $v$ is the originator and is denoted by $b(v, G)$. The broadcast time of a given graph $G$, is the maximum broadcast time from any originator in $G$, formally $b(G) = max_{v \in V(G)}\{b(v, G)\}$. A broadcast scheme for an originator $v$ that uses $b(v, G)$ time units is called an optimal broadcast scheme. Obviously, by the assumption (3), the number of informed vertices after each time unit can at most be doubled. Meaning, in general, the number of informed vertices after time unit $i$ is upper bounded by $2^i$. Therefore, it is easy to see that $b(v, G) \geq \lceil \log n \rceil$, where $n$ is the number of vertices in $G$, which implies that $b(G) \geq \lceil \log n \rceil$.

The general *broadcast time decision problem* is formally defined as follows. Given a graph $G = (V, E)$ with a subset $V_0 \subseteq V$, and a positive integer $k$. Can a message be "broadcast" from the base set $V_0$ to all other vertices in time $k$, i.e., is there a sequence $V_0, E_1, V_1, E_2, V_2, \cdots, E_k, V_k$ such that each $V_i \subseteq V$, each $E_i \subseteq E$ $(1 \leq i \leq k)$, $V_k = V$, and, for $1 \leq i \leq k$, (1) each edge in $E_i$ has exactly one endpoint in $V_{i-1}$, (2) no two edges in $E_i$ share a common endpoint, and (3) $V_i = V_{i-1} \cup \{v : \{u, v\} \in E_i\}$? Here $k$ is the total broadcast time, $V_i$ is the set of informed vertices at round $i$, and $E_i$ is the set of edges used for placing calls at round $i$. It is obvious that when $|V_0| = 1$ then this problem becomes the single source broadcast problem of determining $b(v, G)$ for an arbitrary vertex $v$ in an arbitrary graph $G$.

Generally, the broadcast time decision problem in an arbitrary graph is NP-complete [8,21]. Moreover, the minimum broadcast time problem was proved to be NP-complete even for some restricted graph families, such as 3-regular planar graphs [19], and series-parallel graphs [22]. The study of the parameterized complexity of the broadcast time problem was initiated in [6]. There is a very limited number of graph families, for which an exact algorithm with polynomial time complexity is known for the broadcast time problem. Exact linear time algorithms are available for the broadcast time problem in trees [20,21], in connected graphs with only one cycle (unicyclic graphs) [13,14], in necklace graphs (chain of rings) [11], in $k$-restricted cactus graphs [23], in fully connected trees [9], and in Harary-like graphs [2,3]. For a more detailed introduction to broadcasting and related problems, we refer the reader to [7,12,15,16].

In this paper, we pose a new problem of finding an optimal permutation of two given lists, such that an element-wise subtraction of the two from a third list minimizes the maximum element of the resulting list. We also discuss its application to solving the broadcast time problem on $k$-path graphs, which are a subfamily of 2-connected series-parallel graphs.

The rest of this paper is organized as follows. In Sect. 2, we discuss some previous results on broadcasting in $k$-path graphs. Further, in Sect. 3, we define 3-List-Sub problem and prove some properties of the problem and some of its variations. In Sect. 4, We demonstrate a close connection between the proposed problem and broadcasting in $k$-path graphs, enabling us to devise an exact polynomial-time algorithm on a subfamily of $k$-path graphs. Finally, we will conclude the paper in Sect. 5.

## 2 *k*-Path Graphs

In this section, we discuss a simple subfamily of 2-connected series-parallel graphs that are usually referred to as *k-path graphs* or *melon graphs*. A $k$-path graph $G = (P_1, P_2, ..., P_k)$ is obtained from a pair of vertices $u$ and $v$, by adding $k \geq 2$ internally vertex-disjoint paths $P_1, P_2, ..., P_k$ between $u$ and $v$. Vertices $u$ and $v$ are called junctions of $G$ (Fig. 1).

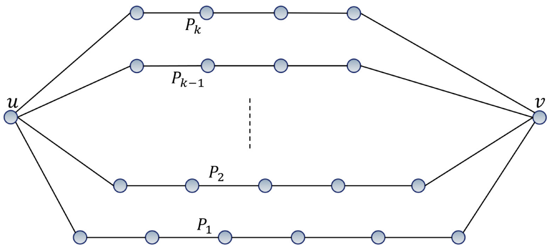


**Fig. 1.** Example of a $k$-path graph.

We assume that paths are indexed in a non-increasing order of their lengths. Formally, $l_1 \geq l_2 \geq ... \geq l_k$, where $l_i$ is the number of vertices on path $P_i$ (excluding $u$ and $v$) for all $1 \leq i \leq k$. Note that since we consider all input graphs to be simple, only $l_k$ can be 0.

To have more understanding of the broadcast time problem in general graphs, it is rather important to understand the properties of graphs that make this problem difficult. One such property is the existence of intersecting cycles. Since $k$-path graphs are one of the simplest graphs that contain intersecting cycles, they are very interesting to research in terms of the broadcast time problem. Moreover, we believe that $k$-path graphs are a boundary family between graphs where broadcasting is computationally easy and where it is difficult.

The broadcast time problem was researched for general series-parallel graphs as well as for subclasses of series-parallel graphs [17,18,22]. In [22], the authors show that the minimum broadcast time problem restricted to graphs with *treewidth* 2 is NP-complete (for an introduction to treewidth we refer the reader to [4,5]). As series-parallel graphs have a treewidth of 2, the result also applies to series-parallel graphs.

In [1], the authors introduce a $(4-\epsilon)$-approximation algorithm for the broadcast time problem in general $k$-path graphs. Additionally, for some particular subclasses of $k$-path graphs, the authors give better approximations or optimal algorithms. The complete set of results introduced in [1] is presented in Table 1. Later, a 2-approximation algorithm for the broadcast time problem in general $k$-path graphs was introduced in [10].

**Table 1.** Summary of known results for $k$-path graphs [1,10]

| Case | Algorithm | Result |
|---|---|---|
| General $k$-path | [10] | 2-approximation |
| $l_j \geq l_{j+1}+2$ and $k \leq l_k+1$ | $S_{path}$ [1] | optimal |
| $l_j = l_{j+1}$ and $k \leq l_k+1$ | $S_{path}$ [1] | optimal |
| $l_j = l_{j+1}+1$ and $k \leq l_k+1$ | $S_{path}$ [1] | $(\frac{4}{3}-\epsilon)$-approximation |
| $l_j = l_{j+1}+1$, $k \leq l_k+1$ and $u$ is the originator | $A_{path}$ [1] | $(\frac{7}{6}-\epsilon)$-approximation |

In [1], the authors also presented several lower bounds and other auxiliary results for the broadcast time problem.

Given a $k$-path graph $G_k$ and junction vertex $u$ authors proved the following.

**Lemma 1** [1]. *There exists a minimum broadcast scheme from the originator $u$ in $G_k$ in which the shortest path $P_k$ is informed in the first time unit.*

Similarly, given an internal vertex $w$ authors proved the following.

**Lemma 2** [1]. *There exists a minimum broadcast scheme from the originator $w$ in $G_k$ in which $w$ first sends the message along a shorter path towards a junction vertex.*

## 3 3-List-Sub Problem

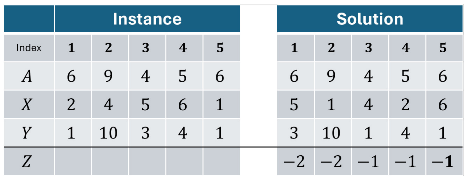
For a list $A$, we refer to the $i^{th}$ element of $A$ as $A[i]$ or $A_i$. Unless otherwise stated, in the context of this section, we assume that indexing of lists starts from 1. Given 3 lists of integers $A$, $X$, and $Y$ of the same size $n$, let *list subtraction* $LS(A, X, Y)$ denote a list of integers $Z$, such that $Z_i = A_i - X_i - Y_i$. Also, let $MLS(A, X, Y) = \max\{Z_i | 1 \leq i \leq n\}$, where $MLS$ stands for *maximum list subtraction*. The goal of the 3 lists subtraction problem (3-List-Sub) is to find permutations $X'$ and $Y'$ of $X$ and $Y$, respectively, that minimize $MLS(A, X', Y')$.

We formally define the general *3 list subtraction* decision problem as follows:

**3 List Subtraction Problem (3-List-Sub):** *Let $A$, $X$, and $Y$ be lists of integers of the same size $n$, and $k$ be a positive integer. Do there exist permutations $X'$ and $Y'$ of $X$ and $Y$, respectively, that meet the following condition?*

$$\max_{1 \le i \le n} \{A_i - X'_i - Y'_i\} \le k$$

An example of 3-List-Sub problem instance and its optimal solution are provided in Fig. 2.



| | Instance | | | | | Solution | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Index | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| $A$ | 6 | 9 | 4 | 5 | 6 | 6 | 9 | 4 | 5 | 6 |
| $X$ | 2 | 4 | 5 | 6 | 1 | 5 | 1 | 4 | 2 | 6 |
| $Y$ | 1 | 10 | 3 | 4 | 1 | 3 | 10 | 1 | 4 | 1 |
| $Z$ | | | | | | −2 | −2 | −1 | −1 | **−1** |

**Fig. 2.** Example of a 3-List-Sub instance and its solution.

For lists $A$, $X$, and $Y$, we denote by $M(A, X, Y)$ the minimum value of $k$, for which 3-List-Sub can be solved on $A$, $X$, and $Y$. In other words, $M(A, X, Y)$ is the value of minimum $MLS$ out of all permutations of $X$ and $Y$.

It is easy to see that, regardless of the selected permutations of $X$ and $Y$, the sum of the elements in the list subtraction is the same. Moreover, the theoretical minimum is achieved if all elements in the list subtraction are equal (which may not always be possible). Hence, the following lemma is trivial.

**Lemma 3.** *For any lists $A$, $X$, and $Y$ of size $n$,*

$$M(A, X, Y) \ge \frac{\sum_{1 \le i \le n}(A_i - X_i - Y_i)}{n}$$

*Moreover, if all the elements in the list subtraction are equal then the solution is optimal.*

Particularly, we are interested in a variation of 3-List-Sub, referred to as 3-C-List-Sub, where both $X$ and $Y$ comprise consecutive numbers, i.e. $X_i = X_{i-1} + 1$ and $Y_i = Y_{i-1} + 1$, for all $2 \le i \le n$.

## 3.1 Equivalency of 3-List-Sub instances

We will denote with $\Pi : \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, n\}$, a mapping function defining the relation between the indices of a list and its permutation. We will sometimes use the notation $\Pi(X)$ to refer to the list that results after $\Pi$ is applied on the indices of $X$. We say that a pair of permutation functions $(\Pi_1, \Pi_2)$ is an optimal solution to a 3-List-Sub problem on lists $A$, $X$, and $Y$, if applying $\Pi_1$ and $\Pi_2$ on $X$ and $Y$, respectively, results in permutation lists with minimum $MLS$.

**Lemma 4.** *Let $A$, $X$, $Y$, $B$, and $C$ be lists of $n$ integers, where $X$, $Y$, $B$ and $C$ consist of consecutive integers and $B_1 = C_1 = 1$. If a pair of permutation functions $(\Pi_1, \Pi_2)$ is an optimal solution to the* 3-List-Sub *instance on $A$, $X$, and $Y$, then $(\Pi_1, \Pi_2)$ is also an optimal solution to the* 3-List-Sub *instance on $A$, $B$, and $C$.*

*Proof.* Let $X'$ and $B'$ denote the lists that result when $\Pi_1$ is applied on $X$ and $B$, respectively. More formally, $X'_j = X_i$ and $B'_j = B_i$, when $j = \Pi_1(i)$, $1 \leq i \leq n$. Similarly, let $Y'$ and $C'$ denote the lists that result when $\Pi_2$ is applied on $Y$ and $C$, respectively. Consider the lists $Z = LS(A, X', Y')$ and $D = LS(A, B', C')$. Since, $X$, $Y$, $B$, and $C$ all comprise consecutive integers and $B_1 = C_1 = 1$, the following is easy to see for any $1 \leq i \leq n$.

- $X'_j - B'_j = X_i - B_i = (X_1 + i - 1) - i = X_1 - 1$, where $j = \Pi_1(i)$.
- $Y'_j - C'_j = Y_i - C_i = (Y_1 + i - 1) - i = Y_1 - 1$, where $j = \Pi_2(i)$.
- $D_j - Z_j = (A_j - B'_j - C'_j) - (A_j - X'_j - Y'_j) = X'_j - B'_j + Y'_j - C'_j = X_1 + Y_1 - 2$

In other words, all elements in $Z$ and $D$ with the same index differ by a constant. Assuming $MLS(A, X', Y')$ is met at some index $h$, $MLS(A, B', C')$ should also be met at index $h$. Otherwise, it would contradict the constant difference. Thus, $z - d = X_1 + Y_1 - 1$, where $z = MLS(A, X', Y')$ and $d = MLS(A, B', C')$.

Assume, by contradiction, that $(\Pi_1, \Pi_2)$ is not an optimal solution to the 3-List-Sub instance on $A$, $B$, and $C$. Let $(\Pi'_1, \Pi'_2)$ be an optimal solution to the same instance, meaning $MLS(A, \Pi'_1(B), \Pi'_2(C)) = M(A, B, C) = d' < d$. Let $z' = MLS(A, \Pi'_1(X), \Pi'_2(Y))$ be the outcome when $(\Pi'_1, \Pi'_2)$ is applied to the instance on $A$, $X$, and $Y$. Based on the above observations, $z' - d' = X_1 + Y_1 - 1$, which results in the following contradiction;

$$z' = d' + X_1 + Y_1 - 1 < d + X_1 + Y_1 - 1 < z$$

□

## 3.2 Optimal Solution to Restricted 3-C-List-Sub

Given an instance of 3-C-List-Sub on lists $A$, $X$, and $Y$ of size $n$, where all the lists consist of consecutive integers, we are going to design optimal permutation functions. By Lemma 4, we can assume that $X_1 = Y_1 = 1$, since the permutation functions will also apply to other cases. We will give the optimal permutations based on the parity of $n$.

**Odd *n*** Let $p = \frac{n-1}{2}$. Consider the following permutations $X'$ and $Y'$ of $X$ and $Y$, respectively.

$$\begin{cases} X'_j = X_{2j} = 2j & \text{where } 1 \leq j \leq p \\ X'_{1+p+j} = X_{2j+1} = 2j + 1 & \text{where } 0 \leq j \leq p \end{cases} \tag{1}$$

$$\begin{cases} Y'_j = Y_{p-j+1} = p - j + 1 & \text{where } 1 \leq j \leq p \\ Y'_{1+p+j} = Y_{n-j} = n - j & \text{where } 0 \leq j \leq p \end{cases} \tag{2}$$

**Theorem 1.** *$X'$ and $Y'$ are optimal permutations for the* 3-C-List-Sub *instance on $A$, $X$, and $Y$ of odd size $n$, and $M(A,X,Y) = A_1 - p - 2$.*

*Proof.* Following the definitions of the permutations, we are going to calculate $Z = LS(A, X', Y')$, considering each half of the list $Z$ separately.

– For $1 \leq j \leq p$

$$Z_j = A_j - X'_j - Y'_j = (A_1 + j - 1) - (2j) - (p - j + 1) = A_1 - p - 2 \quad (3)$$

– For $0 \leq j \leq p$, $i = 1 + p + j$

$$\begin{aligned} Z_i = A_i - X'_i - Y'_i &= (A_1 + 1 + p + j - 1) - (2j + 1) - (n - j) \\ &= A_1 + p - n - 1 = A_1 + p - (2p + 1) - 1 = A_1 - p - 2 \end{aligned} \quad (4)$$

Since all elements in $Z$ are equal, by Lemma 3, $X'$ and $Y'$ are optimal permutations for the 3-C-List-Sub instance on $A$, $X$, and $Y$. Moreover, $M(A,X,Y) = A_1 - p - 2$. □

**Even *n*** Let $p = \frac{n}{2}$. Consider the following permutations $X'$ and $Y'$ of $X$ and $Y$, respectively.

$$\begin{cases} X'_j = X_{2j} = 2j & \text{where } 1 \leq j \leq p \\ X'_{1+p+j} = X_{2j+1} = 2j + 1 & \text{where } 0 \leq j < p \end{cases} \quad (5)$$

$$\begin{cases} Y'_j = Y_{p-j+1} = p - j + 1 & \text{where } 1 \leq j \leq p \\ Y'_{1+p+j} = Y_{n-j} = n - j & \text{where } 0 \leq j < p \end{cases} \quad (6)$$

**Theorem 2.** *$X'$ and $Y'$ are optimal permutations for the* 3-C-List-Sub *instance on $A$, $X$, and $Y$ of even size $n$, and $M(A,X,Y) = A_1 - p - 2$.*

*Proof.* Following the definitions of the permutations, we are going to calculate $Z = LS(A, X', Y')$, considering each half of the list $Z$ separately.

– For $1 \leq j \leq p$

$$Z_j = A_j - X'_j - Y'_j = (A_1 + j - 1) - (2j) - (p - j + 1) = A_1 - p - 2 \quad (7)$$

– For $0 \leq j < p$, $i = 1 + p + j$

$$\begin{aligned} Z_i = A_i - X'_i - Y'_i &= (A_1 + 1 + p + j - 1) - (2j + 1) - (n - j) \\ &= A_1 + p - n - 1 = A_1 + p - (2p + 1) - 1 = A_1 - p - 2 \end{aligned} \quad (8)$$

Since all elements in $Z$ are equal, by Lemma 3, $X'$ and $Y'$ are optimal permutations for the 3-C-List-Sub instance on $A$, $X$, and $Y$. Moreover, $M(A,X,Y) = A_1 - p - 2$. □

## 4 Reducing Broadcast Time Problem on *k*-Path Graphs

Let $G = (P_1, P_2, ..., P_k)$ be a $k$-path graph with junction vertices $u$ and $v$. Recall that we assume paths are sorted in a non-increasing order of their lengths, i.e. $l_1 \geq l_2 \geq ... \geq l_k \geq 0$.

**Theorem 3.** *Consider the case when $u$ is the originator. We reduce the minimum broadcast time problem on $k$-path graphs to* 3-C-List-Sub *on the following lists: $A = \{l_{k-1}, l_{k-2}, \dots, l_2, l_1\}$, $X = \{1, 2, 3, \dots, k-1\}$, $Y = \{1, 2, 3, \dots, k-1\}$ If $b(u, G) \geq l_k + k$, the permutation functions of $X$ and $Y$ that optimally solve* 3-C-List-Sub *instance on $A$, $X$, and $Y$, also induce optimal broadcasting order of junctions $u$ and $v$ in graph $G$.*

*Proof.* By Lemma 1, $u$ passes the message along the path $P_k$ to $v$ in the first time unit. Hence, $v$ will be informed in time unit $t_v = l_k + 1$. Consider the time unit $t_v + k - 1$.

- Since $v$ can start informing its neighbors from time unit $t_v + 1$, by time unit $t_v + k - 1$, all of its $k - 1$ uninformed neighbors (other than on path $P_k$) can potentially be informed. Moreover, the path that was informed on time $t_v + i$ can have $k - i$ informed vertices, for any $1 \leq i \leq k - 1$. This means $v$ can contribute to informing $\{1, 2, \dots, k - 1\}$ vertices on some paths, depending on the order of calls placed by $v$.
- Similarly, $u$ starts informing its neighbors from time unit 2 (excluding path $P_k$). Hence, by time unit $t_v + k - 1$, the path that was informed by $u$ on time $1 + i$ can have $t_v + k - 1 - i$, for any $1 \leq i \leq k - 1$. Meaning $u$ can contribute to informing $\{t_v, t_v + 1, \dots, t_v + k - 1\}$ vertices on some paths, depending on the order of calls placed by $v$.

Starting from time unit $t_v + k$, neither $u$ nor $v$ can affect the broadcasting process. In each time unit after that, one or two vertices on each path can be informed, until the broadcasting is finished. Hence, the decisive factor of the broadcast time is the order of calls placed by $u$ and $v$.

Consider the instance of 3-C-List-Sub on lists $A, B$, and $C$, where $A = \{l_1, l_2, \dots, l_{k-2}, l_{k-1}\}$, $B = \{1, 2, 3, \dots, k-1\}$, $C = \{t_v, t_v+1, \dots, t_v+k-2\}$. Let $B'$ and $C'$ be any permutations of $B$ and $C$, respectively. Let $D = LS(A, B', C')$ and $d = MLS(A, B', C')$. Note that permutations $B'$ and $C'$ uniquely define the order of calls (broadcast scheme $S$) placed by $v$ and $u$, respectively. Whereas $D_i$, for any $1 \leq i \leq k - 1$, corresponds to the number of uninformed vertices on path $P_i$ in time unit $t_v + k$, if $v$ and $u$ followed the broadcast order defined by $B'$ and $C'$. Meaning $d$ is the highest number of uninformed vertices on any path. Assume, WLOG, $d = D_j$ for some $1 \leq j \leq k - 1$. Since path $P_j$ has the highest number of uninformed vertices $d$, there are two possible cases.

- If $d \geq 0$: In this case, starting from the time unit $t_v + k$, up to two vertices can be informed on path $P_j$. This means that the broadcasting will be over after $\lceil \frac{d}{2} \rceil$ time units, hence, the following broadcast time will be achieved.

$$b(S) = t_v + k - 1 + \left\lceil \frac{d}{2} \right\rceil = l_k + k + \left\lceil \frac{d}{2} \right\rceil \tag{9}$$

- If $d < 0$: This case implies that by time unit $t_v + k - 1$, none of the paths had any uninformed vertices, i.e. broadcast was finished and $b(S) < l_k + 1 + k - 1 = l_k + k$. Which makes this case inapplicable to this theorem. However, we can note that $P_j$ was fully informed
  - at least $\lceil \frac{d}{2} \rceil$ time units before $t_v + k - 1$, if it was informed by both $u$ and $v$,
  - and at most $d$ time units before $t_v + k - 1$, if it was informed by only one of $u$ and $v$.

Thus, by Eq. (9), to minimize $B(S)$, $d$ should be equal to $M(A, B, C)$.

$$b(u, G) = l_k + k + \left\lceil \frac{M(A, B, C)}{2} \right\rceil \tag{10}$$

On the other hand, by Lemma 4, the 3-C-List-Sub instance on $A$, $B$, and $C$ has the same optimal permutations as the instance on $A$, $X$, and $Y$. □

**Corollary 1**

$$\begin{cases} b(u, G) = l_k + k + \left\lceil \frac{M(A,B,C)}{2} \right\rceil & \textit{if } b(u, G) \geq l_k + k \\ l_k + k + M(A, B, C) \leq b(u, G) \leq l_k + k + \left\lceil \frac{M(A,B,C)}{2} \right\rceil & \textit{if } b(u, G) < l_k + k \end{cases}$$

Another case is when a vertex $w$ on path $P_m$ is the originator. Let $d$ be the length of the path $\overline{wu}$, and hence, $l_m + 1 - d$ be the length of the path $\overline{wv}$. Assume, WLOG, that $d \leq l_m + 1 - d$. Also, let $\tau(m) = l_m + 2 - 2d$. Recall that when $\tau(m) < l_k + 2$, the vertex $v$ will be informed via the direct path from $w$, and hence, $t_v = d + \tau(m)$. We reduce the minimum broadcast time problem on $k$-path graphs to 3-C-List-Sub on the following lists.

1. $A = \{l_k, l_{k-1}, \ldots, l_{m+2}, l_{m+1}, l_{m-1}, l_{m-2}, \ldots, l_2, l_1\}$,
2. $X = \{1, 2, 3, \ldots, k - 1\}$, $Y = \{1, 2, 3, \ldots, k - 1\}$,
3. $B = \{1, 2, 3, \ldots, k - 1\}$, $C = \{\tau(m) + 1, \tau(m) + 2, \ldots, \tau(m) + k - 1\}$.

Let $h = d + \tau(m) + k - 1$. The below claims are proved analogous to Theorem 3 and Corollary 1.

**Theorem 4.** *If $b(w, G) \geq h$, the permutation functions of $X$ and $Y$ that optimally solve* 3-C-List-Sub *instance on $A$, $X$, and $Y$, also induce optimal broadcasting order of junctions $u$ and $v$ in graph $G$.*

**Corollary 2**

$$\begin{cases} b(w, G) = h + \left\lceil \frac{M(A,B,C)}{2} \right\rceil & \textit{if } b(w, G) \geq h \\ h + M(A, B, C) \leq b(w, G) \leq h + \left\lceil \frac{M(A,B,C)}{2} \right\rceil & \textit{if } b(w, G) < h \end{cases}$$

### 4.1 Exact Broadcasting on Restricted *k*-Path Graphs

Let $G = (P_1, P_2, \ldots, P_k)$ be a $k$-path graph, where $l_i = l_{i+1} + 1$ for any $1 \leq i \leq k - 2$. The length of path $l_k$ is arbitrary. According to Table 1, the problem of finding an exact broadcasting algorithm for such $k$-path graphs remains open. However, combining the contributions of Sects. 3.2 and 4 results in an exact broadcast algorithm. By Theorem 3, the exact order of calls placed by $u$ and $v$ corresponds to the optimal permutations in the reduced 3-C-List-Sub instance. This means that, in the induced algorithm, both $u$ and $v$ should follow the reverse order of permutation lists described in Eqs. (1), (2), (5), and (6). An example visualizing this behavior is portrayed in Fig. 3. Algorithm 1 formally presents the broadcasting order.

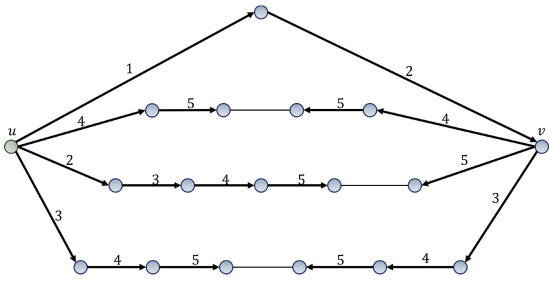


**Fig. 3.** Example of the broadcast scheme in Algorithm 1.

**Algorithm 1.** Broadcasting in restricted $k$-path graphs

```
    Input A $k$-path graph $G = (P_1, P_2, ..., P_k)$, vertices $u$ and $v$, and an originator $u$
    Output Exact broadcast scheme for graph $G$ and the originator $u$
 1: procedure ExactBroadcasting
 2:     In the first time, unit $u$ passes the message along the path $P_k$
 3:     $n \leftarrow k - 1$
 4:     if $n$ is odd then
 5:         $p \leftarrow \frac{n-1}{2}$
 6:         for $1 \le j \le p$ do
 7:             $u$ informs its neighbor on path $P_{k-j}$, on time $n - p + j$
 8:             $v$ informs path $P_{k-j}$, $n - 2j + 1$ time units after being informed
 9:         end for
10:         for $0 \le j \le p$ do
11:             $u$ informs its neighbor on path $P_{k-j-p-1}$, on time $j + 1$
12:             $v$ informs path $P_{k-j-p-1}$, $n - 2j$ time units after being informed
13:         end for
14:     else
15:         $p \leftarrow \frac{n}{2}$
16:         for $1 \le j \le p$ do
17:             $u$ informs its neighbor on path $P_{k-j}$, on time $n - p + j$
18:             $v$ informs path $P_{k-j}$, $n - 2j + 1$ time units after being informed
19:         end for
20:         for $0 \le j < p$ do
21:             $u$ informs its neighbor on path $P_{k-j-p-1}$, on time $j + 1$
22:             $v$ informs path $P_{k-j-p-1}$, $n - 2j$ time units after being informed
23:         end for
24:     end if
25: end procedure
```

## 5 Conclusion and Future Work

In this paper, we introduced a new problem that we refer to as the 3 List Subtraction Problem (3-List-Sub). The goal of the problem is to find permutations of two of the three given lists, such that element-wise subtraction of these two lists from the first list results in a list with the lowest maximum element value. We devised an optimal solution to a restricted subproblem to 3-List-Sub, helping us to find an exact broadcasting algorithm on restricted $k$-path graphs with consecutive path lengths. Additionally, for $k$-path graphs with significantly larger broadcast times of the junctions, we showed that the length of the shortest path does not affect the broadcast order of the junctions. To the best of our knowledge, 3-List-Sub problem and its potential generalizations (e.g. to $k$ different paths) were not previously studied. Hence, we believe that it can be important and interesting to study the NP-hardness of 3-List-Sub problem. When it comes to broadcasting on $k$-path graphs, the problem of designing a polynomial-time exact broadcasting algorithm for arbitrary $k$-path graphs remains open.

## References

1. Bhabak, P., Harutyunyan, H.A.: Approximation algorithm for the broadcast time in k-path graph. J. Interconnection Netw. **19**(04), 1950006 (2019)
2. Bhabak, P., Harutyunyan, H.A., Kropf, P.: Efficient broadcasting algorithm in Harary-like networks. In: 2017 46th International Conference on Parallel Processing Workshops (ICPPW), pp. 162–170 (2017)
3. Bhabak, P., Harutyunyan, H.A., Tanna, S.: Broadcasting in Harary-like graphs. In: 2014 IEEE 17th International Conference on Computational Science and Engineering (CSE), pp. 1269–1276 (2014)
4. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. **209**(1–2), 1–45 (1998)
5. Bodlaender, H.L., Koster, A.M.: Combinatorial optimization on graphs of bounded treewidth. Comput. J. **51**(3), 255–269 (2008)
6. Fomin, F.V., Fraigniaud, P., Golovach, P.A.: Parameterized complexity of broadcasting in graphs. Theor. Comput. Sci. **997**, 114508 (2024)
7. Fraigniaud, P., Lazard, E.: Methods and problems of communication in usual networks. Discret. Appl. Math. **53**(1), 79–133 (1994)
8. Garey, M.R., Johnson, D.S.: Computers and intractability. A guide to the theory of NP-Completeness (1983)
9. Gholami, S., Harutyunyan, H.A., Maraachlian, E.: Optimal broadcasting in fully connected trees. J. Interconnection Netw. **23**(01), 2150037 (2023)
10. Harutyunyan, H.A., Hovhannisyan, N.: Improved approximation for broadcasting in k-path graphs. In: International Conference on Combinatorial Optimization and Applications (COCOA), pp. 111–122. Springer, Cham (2023)
11. Harutyunyan, H.A., Hovhannisyan, N., Maraachlian, E.: Broadcasting in chains of rings. In: 2023 Fourteenth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 506–511. IEEE (2023)
12. Harutyunyan, H.A., Liestman, A.L., Peters, J.G., Richards, D.: Broadcasting and gossiping. In: Handbook of Graph Theory, pp. 1477–1494 (2013)

13. Harutyunyan, H.A., Maraachlian, E.: Linear algorithm for broadcasting in unicyclic graphs. In: International Computing and Combinatorics Conference (COCOON), pp. 372–382. Springer, Cham (2007)
14. Harutyunyan, H.A., Maraachlian, E.: On broadcasting in unicyclic graphs. J. Comb. Optim. **16**(3), 307–322 (2008)
15. Hedetniemi, S.M., Hedetniemi, S.T., Liestman, A.L.: A survey of gossiping and broadcasting in communication networks. Networks **18**(4), 319–349 (1988)
16. Hromkovič, J., Klasing, R., Monien, B., Peine, R.: Dissemination of information in interconnection networks (broadcasting & gossiping). In: Combinatorial Network Theory, pp. 125–212. Springer, Cham (1996)
17. Kortsarz, G., Peleg, D.: Approximation algorithms for minimum-time broadcast. SIAM J. Discret. Math. **8**(3), 401–427 (1995)
18. Marathe, M.V., Ravi, R., Sundaram, R., Ravi, S., Rosenkrantz, D.J., Hunt, H.B., III.: Bicriteria network design problems. J. Algorithms **28**(1), 142–171 (1998)
19. Middendorf, M.: Minimum broadcast time is NP-Complete for 3-regular planar graphs and deadline 2. Inf. Process. Lett. **46**(6), 281–287 (1993)
20. Proskurowski: Minimum broadcast trees. IEEE Trans. Comput. **C-30**(5), 363–366 (1981)
21. Slater, P.J., Cockayne, E.J., Hedetniemi, S.T.: Information dissemination in trees. SIAM J. Comput. **10**(4), 692–701 (1981)
22. Tale, P.: Double exponential lower bound for telephone broadcast. CoRR abs/2403.03501 (2024)
23. Čevnik, M., Žerovnik, J.: Broadcasting on cactus graphs. J. Comb. Optim. **33**(1), 292–316 (2017)

# The Power of Second Chance: Personalized Submodular Maximization with Two Candidates

Jing Yuan[1] and Shaojie Tang[2](✉)

[1] Department of Computer Science and Engineering, University of North Texas, Denton, USA

[2] Center for AI Business Innovation, Department of Management Science and Systems, University at Buffalo, Buffalo, USA

tangshaojie@gmail.com

**Abstract.** Most of existing studies on submodular maximization focus on selecting a subset of items that maximizes a *single* submodular function. However, in many real-world scenarios, we might have multiple user-specific functions, each of which models the utility of a particular type of user. In these settings, our goal would be to choose a set of items that performs well across all the user-specific functions. One way to tackle this problem is to select a single subset that maximizes the sum of all of the user-specific functions. Although this aggregate approach is efficient in the sense that it avoids computation of sets for individual functions, it really misses the power of personalization - for it does not allow to choose different sets for different functions. In this paper, we introduce the problem of personalized submodular maximization with two candidate solutions. For any two candidate solutions, the utility of each user-specific function is defined as the better of these two candidates. Our objective is, therefore, to select the best set of two candidates that maximize the sum of utilities of all the user-specific functions. We have designed effective algorithms for this problem. We also discuss how our approach generalizes to multiple candidate solutions, increasing flexibility and personalization in our solution.

## 1 Introduction

A submodular function is defined by its intuitive diminishing returns property: adding an item to a smaller set will increase the return more in comparison with when this happens from a larger set. Such a function is extremely common in various combinatorial optimization problems naturally arising from machine learning, graph theory, economics, and game theory. Most of the work in submodular optimization focuses on selecting a subset of items from a ground set that maximizes a single submodular function. However, in many real-world scenarios, we are confronted with multiple user-specific functions denoted as $f_1, \cdots, f_m : 2^{\Omega} \rightarrow \mathbb{R}_{\geq 0}$. Each of these functions, such as $f_i$, captures the utility corresponding to some user type indexed by $i$. Our main goal will be to maximize the aggregate utility of all the $m$ functions. One trivial way to achieve this would be to compute a solution individually for every single function $f_i$. Unfortunately, this would require to compute and store $m$ solutions, which is infeasible or at least very inefficient if the number of user-specific functions is large. Another way is to look for a *single* feasible solution, denoted as $S \subseteq \Omega$, that maximizes the summation of these $m$ functions, i.e., $\max_{S\subseteq\Omega} \sum_{i\in[m]} f_i(S)$. This problem, also known as the maximization of decomposable submodular functions [8], has been well-studied in the literature and efficient algorithms have been designed for the same. Nevertheless, such an aggregate approach, despite being efficient, is unable to harness the power of personalization. Specifically, it does not provide the flexibility in offering a personalized set for each function.

In our research, we introduce the innovative concept of personalized submodular maximization. Consider a pair of sets $\{S_1, S_2\}$, for each user-specific function $f_i$, we determine its utility based on the better-performing solution among these two candidates, represented as $\max\{f_i(S_1), f_i(S_2)\}$. Mathematically, our problem can be expressed as follows:

$$\max_{S_1,S_2\subseteq\Omega} \sum_{i\in[m]} \max\{f_i(S_1), f_i(S_2)\}$$
$$\text{subject to } |S_1| \leq k, |S_2| \leq k,$$

where $k$ is the size constraint of a feasible solution. In essence, our primary objective is to maximize the combined utility of user-specific functions while maintaining a personalized approach to item selection. An important and practical application of our study is in the context of two-stage optimization. Here, we consider that $f_1, \cdots, f_m$ represent training examples of functions drawn from an unknown distribution, we aim to choose a pair of candidate solutions based on these $m$ functions, ensuring that one of the chosen candidates performs well when faced with a new function from the same distribution.

In this paper, we also discuss the possibility of expanding our approach to accommodate multiple (more than two) candidate solutions. This potential extension would further enhance the flexibility and personalization options within our solution.

## 1.1 Related Work

The problem of submodular maximization has received considerable attention in the literature [3,4,10,11,13]. For example, one of the most well-established results is that a simple greedy algorithm achieves a tight approximation ratio of $(1-1/e)$ for maximizing a single monotone submodular function subject to cardinality constraints [7]. Since most datasets are so big nowadays, several works were devoted to reducing the running time to maximize a submodular function. Examples include the development of accelerated greedy algorithms [5] and streaming algorithms [1]. All of these works, however, focus on finding a single set that maximizes a submodular function. In contrast, our goal is to identify a pair of candidates that maximizes the sum of the better-performing solution between them. This presents a unique challenge, as the resulting objective function is no longer submodular. Consequently, existing results on submodular optimization cannot be directly applied to our study.

Our work is closely related to the field of two-stage submodular optimization [2,6,9,12], in which the key objective is to find a smaller ground set from a large one. This reduction should be designed in such a way that choosing the items from the small set guarantees approximately the same performance as choosing items from the original large set for a variety of submodular functions. This aligns with our objective of seeking two initial solutions that cut down on computational effort in optimization with a new function. However, problem formulations between our studies are largely different despite sharing the same objective. Thereby, new methodologies should be developed to cope with the distinctive challenges presented in our research. Moreover, note that in the traditional framework of two-stage submodular optimization, once a reduced ground set is computed, further optimization based on this reduced set usually involves algorithms with possibly high time complexity, such as the greedy algorithm. In contrast, our personalized optimization model requires only a comparison between the performance of two candidate solutions, significantly reducing the computational burden in the second stage.

## 2 Problem Formulation

Our problem involves an input set of $n$ items denoted as $\Omega$, and a collection of $m$ submodular functions, namely, $f_1, \cdots, f_m : 2^{\Omega} \rightarrow \mathbb{R}_{\geq 0}$. To clarify, the notation $\Delta_i(x, A)$ denotes the marginal gain of adding item $x$ to set $A$ with respect to the function $f_i$. That is, $\Delta_i(x, A) = f_i(\{x\} \cup A) - f_i(A)$. Specifically, a function $f_i$ is considered submodular if and only if $\Delta_i(x, A) \geq \Delta_i(x, A')$ holds for any two sets $A$ and $A'$ where $A \subseteq A' \subseteq \Omega$ and for any item $x \in \Omega$ such that $x \notin A'$.

Our aim is to select a pair of candidate solutions, $S_1$ and $S_2$, and the utility of each user-specific function is determined by the superior solution among these two candidates. These subsets should provide good performance across all $m$ functions when we are limited to choosing solutions from either $S_1$ or $S_2$. Formally,

$$\boxed{\begin{array}{l}\textbf{P.0}\ \max_{S_1, S_2 \subseteq \Omega} \sum_{i \in [m]} \max\{f_i(S_1), f_i(S_2)\} \\ \text{subject to } |S_1| \leq k, |S_2| \leq k,\end{array}}$$

where $k$ is the size constraint of a feasible solution.

A straightforward approach to solving **P.0** is to transform it into a standard set selection problem. Specifically, we can introduce a ground set $\mathcal{U} = \{(i, j) \mid i \in \Omega, j \in \{1, 2\}\}$. Here, selecting an element $(i, j) \in \mathcal{U}$ corresponds to placing item $i$ in set $S_j$ in our original problem. Let $x_{ij}$ be a binary decision variable representing the selection of $(i, j)$, such that $x_{ij} = 1$ if and only if $(i, j)$ is selected. Then **P.0** is reduced to finding a set of elements from $\mathcal{U}$ such that $\forall i \in \Omega, x_{i1} + x_{i2} = 1$ and $\forall j \in \{1, 2\}, \sum_{i \in \Omega} x_{ij} \leq k$, which represents the intersection of two matroid constraints. Unfortunately, it is straightforward to verify that the utility function defined over $\mathcal{U}$ is not necessarily submodular, even if each individual function $f_i$ is submodular. Hence, existing solutions for submodular maximization subject to two matroid constraints are not directly applicable to our problem.

## 3 Algorithm Design for Constant $m$

We first study the case if the number of functions $m$ is a constant. Before presenting our algorithm, we introduce a new optimization problem **P.1**. The objective of this problem is to partition the $m$ functions into two groups such that the sum of the optimal solutions for these two groups is maximized. Formally,

**P.1**

$$\max_{A,B\subseteq[m]} \Big( \max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in A} f_i(S) + \max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in B} f_i(S) \Big)$$

subject to $B = [m] \setminus A$.

We next show that the optimal solution of **P.1** serves as an upper bound for our original problem.

**Lemma 1.** *Let $OPT_1$ (resp. $OPT_0$) denote the value of the optimal solution of **P.1** (resp. our original problem **P.0**), we have*

$$OPT_1 \geq OPT_0. \tag{1}$$

*Proof:* Assume $S_1^*$ and $S_2^*$ is the optimal solution of **P.0**, we can partition $m$ functions to two groups $A'$ and $B'$ such that every function in $A'$ favors $S_1^*$ and every function in $B'$ favors $S_2^*$. That is,

$$A' = \{i \in [m] \mid f_i(S_1^*) \geq f_i(S_2^*)\}$$

and

$$B' = \{i \in [m] \mid f_i(S_1^*) < f_i(S_2^*)\}.$$

Hence,

$$\begin{aligned} OPT_0 &= \sum_{i\in[m]} \max\{f_i(S_1^*), f_i(S_2^*)\} \\ &= \sum_{i\in A'} \max\{f_i(S_1^*), f_i(S_2^*)\} + \sum_{i\in B'} \max\{f_i(S_1^*), f_i(S_2^*)\} \\ &= \sum_{i\in A'} f_i(S_1^*) + \sum_{i\in B'} f_i(S_2^*) \end{aligned}$$

where the first inequality is by the definition of $OPT_0$, the second equality is by the observation that $A'$ and $B'$ is a partition of $[m]$ and the third equality is by the definitions of $A'$ and $B'$.

Moreover, it is easy to verify that

$$\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A'} f_i(S) + \max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in B'} f_i(S)$$
$$\geq \sum_{i\in A'} f_i(S_1^*) + \sum_{i\in B'} f_i(S_2^*).$$

This is because $|S_1^*| \leq k$ and $|S_2^*| \leq k$. It follows that

$$OPT_0 = \sum_{i\in A'} f_i(S_1^*) + \sum_{i\in B'} f_i(S_2^*)$$
$$\leq \max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A'} f_i(S) + \max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in B'} f_i(S).$$

Therefore,

$$OPT_1 =$$
$$\max_{A,B\subseteq[m]}\big(\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A} f_i(S) + \max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in B} f_i(S)\big)$$
$$\geq \max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A'} f_i(S) + \max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in B'} f_i(S)$$
$$\geq \sum_{i\in A'} f_i(S_1^*) + \sum_{i\in B'} f_i(S_2^*) = OPT_0.$$

This finishes the proof of this lemma. □

Now, we present our algorithm, called Enumeration-based Algorithm, which is listed in Algorithm 1. Our approach involves enumerating all possible partitions of $[m]$. For each partition, denoted as $A$ and $B$, we utilize a state-of-the-art algorithm to solve two subproblems:

$$\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A} f_i(S)$$

and

$$\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in B} f_i(S).$$

This results in obtaining two sets, $C_1$ and $C_2$, respectively. Finally, we return the best pair of sets as the solution for our original problem **P.0**.

Since the number of functions $m$ is a constant, the maximum number of possible partitions we must enumerate is at most $O(2^m)$, which is also a constant. As long as $\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A} f_i(S)$ and $\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in B} f_i(S)$ can be solved in polynomial time, the Enumeration-based Algorithm is a polynomial time algorithm. Next we provide an approximation ratio of Algorithm 1.

**Lemma 2.** *Assuming the existence of $\alpha$-approximation algorithms for*

$$\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A} f_i(S)$$

**Algorithm 1.** Enumeration-based Algorithm

```
1: $S_1 \leftarrow \emptyset, S_2 \leftarrow \emptyset$
2: for $A \subseteq [m]$ do
3:     $B \leftarrow [m] \setminus A$
4:     $C_1 \leftarrow \alpha$-approximation solution of
           $$\max_{S \subseteq \Omega: |S| \leq k} \sum_{i \in A} f_i(S)$$
5:     $C_2 \leftarrow \alpha$-approximation solution of
           $$\max_{S \subseteq \Omega: |S| \leq k} \sum_{i \in B} f_i(S)$$
6:     if
           $$\sum_{i \in [m]} \max\{f_i(C_1), f_i(C_2)\} \geq \sum_{i \in [m]} \max\{f_i(S_1), f_i(S_2)\}$$
       then
7:         $(S_1, S_2) \leftarrow (C_1, C_2)$
8: return $S_1, S_2$
```

*for any $A \subseteq [m]$, our* Enumeration-based Algorithm *(Algorithm 1) provides an $\alpha$-approximation solution for **P.0**.*

*Proof:* Assuming that $A^*$ and $B^*$ represent the optimal solution for **P.1**, let us consider the round of our algorithm where it enumerates the partition of $A^*$ and $B^*$. In this round, we denote the solutions obtained as $C_1$ and $C_2$. Given that there exist $\alpha$-approximation algorithms for $\max_{S \subseteq \Omega: |S| \leq k} \sum_{i \in A} f_i(S)$ for any $A \subseteq [m]$, by adopting this algorithm as a subroutine, we have

$$\sum_{i \in A^*} f_i(C_1) \geq \alpha \max_{S \subseteq \Omega: |S| \leq k} \sum_{i \in A^*} f_i(S)$$

and

$$\sum_{i \in B^*} f_i(C_2) \geq \alpha \max_{S \subseteq \Omega: |S| \leq k} \sum_{i \in B^*} f_i(S).$$

Hence,

$$\begin{aligned}
&\sum_{i \in [m]} \max\{f_i(C_1), f_i(C_2)\} \geq \sum_{i \in A^*} f_i(C_1) + \sum_{i \in B^*} f_i(C_2) \\
&\geq \alpha\big( \max_{S \subseteq \Omega: |S| \leq k} \sum_{i \in A^*} f_i(S) + \max_{S \subseteq \Omega: |S| \leq k} \sum_{i \in B^*} f_i(S)\big) \\
&= \alpha OPT_1
\end{aligned}$$

where the equality is by the assumption that $A^*$ and $B^*$ represent the optimal solution for **P.1**.

This, together with Lemma 1, implies that

$$\sum_{i\in[m]} \max\{f_i(C_1), f_i(C_2)\} \geq \alpha OPT_1 \geq \alpha OPT_0. \tag{2}$$

This lemma is a consequence of the above inequality and the fact that the final solution obtained by our algorithm is at least as good as $\sum_{i\in[m]} \max\{f_i(C_1), f_i(C_2)\}$. □

Observe that if all $f_i$ are monotone and submodular functions, then there exists $(1-1/e)$-approximation algorithms for $\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A} f_i(S)$ for any $A \subseteq [m]$. Therefore, by substituting $\alpha = 1 - 1/e$ into Lemma 2, we obtain the following theorem.

**Theorem 1.** *Assume all $f_i$ are monotone and submodular functions, Enumeration-based Algorithm (Algorithm 1) provides an $(1-1/e)$-approximation solution for **P.0**.*

## 4 Algorithm Design for Large *m*

When dealing with a large value of $m$, relying on an enumeration-based approach can become impractical. In this section, we introduce a Sampling-based Algorithm, outlined in Algorithm 2, that provides provable performance bounds. Instead of exhaustively enumerating all possible partitions of $[m]$, we examine $T$ *random* partitions. For each partition, we follow the same procedure as in Algorithm 1 to compute two candidate solutions. Specifically, for each sampled partition, we employ a state-of-the-art $\alpha$-approximation algorithm to solve two subproblems. Ultimately, we return the best pair of sets as the final solution.

In the following two lemmas, we provide two performance bounds for Algorithm 2. The first bound is independent of the number of samples $T$; thus, it holds even if $T = 1$. The second bound depends on $T$, increasing as $T$ increases.

**Lemma 3.** *Assuming the existence of $\alpha$-approximation algorithms for*

$$\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A} f_i(S)$$

*for any $A \subseteq [m]$, our Sampling-based Algorithm (Algorithm 2) provides an $\alpha/2$-approximation solution for **P.0**.*

*Proof:* We first recall some notations form the proof of Lemma 1. Assume $S_1^*$ and $S_2^*$ is the optimal solution of **P.0**, we partition all $m$ functions to two groups $A'$ and $B'$ such that every function in $A'$ favors $S_1^*$ and every function in $B'$ favors $S_2^*$. That is,

$$A' = \{i \in [m] \mid f_i(S_1^*) \geq f_i(S_2^*)\}$$

and

$$B' = \{i \in [m] \mid f_i(S_1^*) < f_i(S_2^*)\}.$$

**Algorithm 2.** Sampling-based Algorithm

```
1: $S_1 \leftarrow \emptyset, S_2 \leftarrow \emptyset, T$
2: for $t \in [T]$ do
3:     Randomly sample a subset of functions $A \subseteq [m]$
4:     $B \leftarrow [m] \setminus A$
5:     $C_1 \leftarrow \alpha$-approximation solution of
           $$\max_{S \subseteq \Omega : |S| \leq k} \sum_{i \in A} f_i(S)$$
6:     $C_2 \leftarrow \alpha$-approximation solution of
           $$\max_{S \subseteq \Omega : |S| \leq k} \sum_{i \in B} f_i(S)$$
7:     if
           $$\sum_{i \in [m]} \max\{f_i(C_1), f_i(C_2)\} \geq \sum_{i \in [m]} \max\{f_i(S_1), f_i(S_2)\}$$
       then
8:         $(S_1, S_2) \leftarrow (C_1, C_2)$
9: return $S_1, S_2$
```

Without loss of generality, we assume that $\sum_{i \in A'} f_i(S_1^*) \geq \sum_{i \in B'} f_i(S_2^*)$, implying that $\sum_{i \in A'} f_i(S_1^*) \geq OPT_0/2$. Now, let us consider any arbitrary partition sample denoted as $A$ and $B$, generated by our algorithm, we have

$$\max_{S \subseteq \Omega : |S| \leq k} \sum_{i \in A} f_i(S) + \max_{S \subseteq \Omega : |S| \leq k} \sum_{i \in B} f_i(S)$$
$$\geq \sum_{i \in A} f_i(S_1^*) + \sum_{i \in B} f_i(S_1^*) \geq \sum_{i \in A'} f_i(S_1^*) \geq OPT_0/2$$

where the first inequality is by the observation that $|S_1^*| \leq k$, the second inequality is by the observation that $A' \subseteq A \cup B$ and the third inequality is by the observation that $\sum_{i \in A'} f_i(S_1^*) \geq OPT_0/2$. Because there exist $\alpha$-approximation algorithms for $\max_{S \subseteq \Omega : |S| \leq k} \sum_{i \in A} f_i(S)$ for any $A \subseteq [m]$, by adopting this algorithm as a subroutine to compute $C_1$ and $C_2$, we have

$$\sum_{i \in A} f_i(C_1) \geq \alpha \max_{S \subseteq \Omega : |S| \leq k} \sum_{i \in A} f_i(S)$$

and

$$\sum_{i \in B} f_i(C_2) \geq \alpha \max_{S \subseteq \Omega : |S| \leq k} \sum_{i \in B} f_i(S).$$

Hence,

$$\sum_{i\in[m]} \max\{f_i(C_1), f_i(C_2)\} \geq \sum_{i\in A} f_i(C_1) + \sum_{i\in B} f_i(C_2)$$
$$\geq \alpha\big(\max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in A} f_i(S) + \max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in B} f_i(S)\big) \geq (\alpha/2)OPT_0$$

where the third inequality is by inequality (3). This finishes the proof of this lemma. □

**Lemma 4.** *Assuming the existence of $\alpha$-approximation algorithms for*

$$\max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in A} f_i(S)$$

*for any $A \subseteq [m]$, our Sampling-based Algorithm (Algorithm 2), after $T$ rounds, provides an $\alpha\gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})$-approximation solution for **P.0** in expectation where $\gamma(T) = 1 - (\frac{1}{2} + \epsilon\frac{e}{\pi})^T$.*

*Proof:* Consider an arbitrary round of our algorithm, and let $A$ and $B$ denote the sampled partition, and let $(C_1, C_2)$ denote the solution returned from this round. Observe that

$$\sum_{i\in[m]} \max\{f_i(C_1), f_i(C_2)\} \geq \sum_{i\in A} f_i(C_1) + \sum_{i\in B} f_i(C_2).$$

Hence, the expected value of $\sum_{i\in[m]} \max\{f_i(C_1), f_i(C_2)\}$, where the expectation is taken over $A, B$, is at least $\mathbb{E}_{A,B}[\sum_{i\in A} f_i(C_1) + \sum_{i\in B} f_i(C_2)]$. Recall that our algorithm runs $T$ rounds and returns the best $(C_1, C_2)$ as the final solution, to prove this lemma, it suffices to show that the expected value of $\sum_{i\in[m]} \max\{f_i(C_1), f_i(C_2)\}$ is at least $\alpha\gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})OPT_0$. To achieve this, we will focus on proving $\mathbb{E}_{A,B}[\sum_{i\in A} f_i(C_1)+\sum_{i\in B} f_i(C_2)] \geq \alpha\gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})OPT_0$. The rest of the proof is devoted to proving this inequality.

First,

$$\begin{aligned}
&\mathbb{E}_{A,B}[\sum_{i\in A} f_i(C_1) + \sum_{i\in B} f_i(C_2)] \\
&\geq \mathbb{E}_{A,B}[\alpha \max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in A} f_i(S) + \alpha \max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in B} f_i(S)] \\
&= \alpha\mathbb{E}_{A,B}[\max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in A} f_i(S) + \max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in B} f_i(S)] \\
&= \alpha\mathbb{E}_{A}[\max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in A} f_i(S)] + \alpha\mathbb{E}_{B}[\max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in B} f_i(S)] \\
&\geq \alpha\mathbb{E}_{A}[\sum_{i\in A} f_i(S_1^*)] + \alpha\mathbb{E}_{B}[\sum_{i\in B} f_i(S_2^*)]. \qquad (3)
\end{aligned}$$

Next, we provide lower bounds for $\mathbb{E}_A[\sum_{i\in A} f_i(S_1^*)]$ and $\mathbb{E}_B[\sum_{i\in B} f_i(S_2^*)]$. Recall that we defined $A' = \{i \in [m] \mid f_i(S_1^*) \geq f_i(S_2^*)\}$ and $B' = \{i \in [m] \mid f_i(S_1^*) < f_i(S_2^*)\}$. Now, for some $\beta \in [0,1]$, let us denote the event as $E$, which occurs when the following condition holds for at least one partition $(A, B)$ that is enumerated by our algorithm: $\frac{|A\cap A'|}{|A'|} \geq \beta$. Because each item of $A'$ is included in $A$ independently with a probability of $1/2$, for any $\beta \in [0,1]$, we have the following:

$$\mathbb{E}_A[\sum_{i\in A} f_i(S_1^*)] \geq \Pr[\mathbf{1}_E = 1] \cdot \beta \sum_{i\in A'} f_i(S_1^*). \tag{4}$$

Consider a random sample $A$ from $[m]$ and observe that each item of $A'$ is included in $A$ independently with a probability of $1/2$, by an "anti-concentration" result on binomial distributions (Lemma 22.2 in [14]), we have

$$\Pr[|A\cap A'| \geq \frac{|A'|}{2} + \epsilon\sqrt{|A'|}] \geq \frac{1}{2} - \epsilon\frac{e}{\pi}.$$

This implies that

$$\Pr[\frac{|A\cap A'|}{|A'|} \geq \frac{1}{2} + \frac{\epsilon}{\sqrt{|A'|}}] \geq \frac{1}{2} - \epsilon\frac{e}{\pi}.$$

Given that $|A'| \leq m$, we further have

$$\Pr[\frac{|A\cap A'|}{|A'|} \geq \frac{1}{2} + \frac{\epsilon}{\sqrt{m}}] \geq \frac{1}{2} - \epsilon\frac{e}{\pi}.$$

If we set $\beta = \frac{1}{2} + \frac{\epsilon}{\sqrt{m}}$, then we can establish a lower bound on the probability of event $E$ occurring after $T$ rounds as follows:

$$\begin{aligned}\Pr[\mathbf{1}_E = 1] &\geq 1 - (1 - \Pr[\frac{|A\cap A'|}{|A'|} \geq \frac{1}{2} + \frac{\epsilon}{\sqrt{m}}])^T \\ &\geq 1 - (\frac{1}{2} + \epsilon\frac{e}{\pi})^T.\end{aligned}$$

This, together with inequalities (4), implies that

$$\mathbb{E}_A[\sum_{i\in A} f_i(S_1^*)] \geq \Pr[\mathbf{1}_E = 1] \cdot \beta \sum_{i\in A'} f_i(S_1^*) \geq (1 - (\frac{1}{2} + \epsilon\frac{e}{\pi})^T) \cdot (\frac{1}{2} + \frac{\epsilon}{\sqrt{m}}) \sum_{i\in A'} f_i(S_1^*).$$

Following the same argument, we can prove that

$$\mathbb{E}_B[\sum_{i\in B} f_i(S_2^*)] \geq (1 - (\frac{1}{2} + \epsilon\frac{e}{\pi})^T) \cdot (\frac{1}{2} + \frac{\epsilon}{\sqrt{m}}) \sum_{i\in B'} f_i(S_2^*). \tag{5}$$

Let $\gamma(T) = 1-(\frac{1}{2}+\epsilon\frac{e}{\pi})^T$. The above two inequalities, together with inequality (3), imply that

$$\begin{aligned}
&\mathbb{E}_{A,B}[\sum_{i\in A} f_i(C_1) + \sum_{i\in B} f_i(C_2)] \geq \alpha\mathbb{E}_A[\sum_{i\in A} f_i(S_1^*)] + \alpha\mathbb{E}_B[\sum_{i\in B} f_i(S_2^*)]\\
&\geq \alpha\gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})\sum_{i\in A'} f_i(S_1^*) + \alpha\gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})\sum_{i\in B'} f_i(S_2^*)\\
&= \alpha\gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})(\sum_{i\in A'} f_i(S_1^*) + \sum_{i\in B'} f_i(S_2^*))\\
&= \alpha\gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})OPT_0.
\end{aligned}$$

This finishes the proof of this lemma. □

By selecting a tighter bound derived from Lemma 3 and Lemma 4, we can establish the following corollary.

**Corollary 1.** *Assuming the existence of $\alpha$-approximation algorithms for*

$$\max_{S\subseteq\Omega:|S|\leq k} \sum_{i\in A} f_i(S)$$

*for any $A \subseteq [m]$, our Sampling-based algorithm (Algorithm 2), after $T$ rounds, provides an $\max\{1/2, \gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})\}\cdot\alpha$-approximation solution for **P.0** in expectation where $\gamma(T) = 1-(\frac{1}{2}+\epsilon\frac{e}{\pi})^T$.*

Observe that if all $f_i$ are monotone and submodular functions, then there exists $(1-1/e)$-approximation algorithms for $\max_{S\subseteq\Omega:|S|\leq k}\sum_{i\in A} f_i(S)$ for any $A \subseteq [m]$. Therefore, substituting $\alpha = 1-1/e$ into Corollary 1, we derive the following theorem.

**Theorem 2.** *Assume all $f_i$ are monotone and submodular functions, Sampling-based algorithm (Algorithm 2), after $T$ rounds, provides an $\max\{1/2, \gamma(T)(\frac{1}{2}+\frac{\epsilon}{\sqrt{m}})\}\cdot(1-1/e)$-approximation solution for **P.0** in expectation where $\gamma(T) = 1-(\frac{1}{2}+\epsilon\frac{e}{\pi})^T$.*

## 5 Discussion on Scenarios with More Than Two Candidates

We next discuss the case if we are allowed to keep $l \geq 2$ candidate solutions. In this extension, our aim is to select $l$ candidate solutions, $S_1, \cdots, S_l$, and the utility of each user-specific function is determined by the superior solution among these candidates. Hence, our problem can be formulated as

$$\max_{S_1,\cdots,S_l \subseteq \Omega} \sum_{i\in[m]} \max\{f_i(S_1),\cdots,f_i(S_l)\}$$

subject to $|S_1| \leq k, \cdots, |S_l| \leq k$ where $k$ is the size constraint of a feasible solution. To tackle this challenge, we can utilize our enumeration-based partition algorithm (Algorithm 1) to find an approximate solution. The procedure involves enumerating all possible ways to partition the set $[m]$ into $l$ groups. For each partition, we employ a state-of-the-art $(1-1/e)$-approximation algorithm to solve the maximization problem within each group. This process generates $l$ sets, and we then choose the best $l$ sets among all partitions as the final solution. By following the same argument used to prove Theorem 1, we can show that this approach guarantees an $(1 - 1/e)$-approximation solution.

## References

1. Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.: Streaming submodular maximization: massive data summarization on the fly. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 671–680 (2014)
2. Balkanski, E., Mirzasoleiman, B., Krause, A., Singer, Y.: Learning sparse combinatorial representations via two-stage submodular maximization. In: International Conference on Machine Learning, pp. 2207–2216. PMLR (2016)
3. Buchbinder, N., Feldman, M., Naor, J., Schwartz, R.: Submodular maximization with cardinality constraints. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1433–1452. SIAM (2014)
4. Gharan, S.O., Vondrák, J.: Submodular maximization by simulated annealing. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1098–1116. SIAM (2011)
5. Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., Krause, A.: Lazier than lazy greedy. In: Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
6. Mitrovic, M., Kazemi, E., Zadimoghaddam, M., Karbasi, A.: Data summarization at scale: a two-stage submodular approach. In: International Conference on Machine Learning, pp. 3596–3605. PMLR (2018)
7. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions-I. Math. Program. **14**(1), 265–294 (1978)
8. Schwartzman, G.: Mini-batch submodular maximization. arXiv preprint arXiv:2401.12478 (2024)
9. Stan, S., Zadimoghaddam, M., Krause, A., Karbasi, A.: Probabilistic submodular maximization in sub-linear time. In: International Conference on Machine Learning, pp. 3241–3250. PMLR (2017)
10. Tang, S.: Beyond pointwise submodularity: non-monotone adaptive submodular maximization in linear time. Theor. Comput. Sci. **850**, 249–261 (2021)
11. Tang, S.: Beyond pointwise submodularity: non-monotone adaptive submodular maximization subject to knapsack and k-system constraints. Theor. Comput. Sci. **936**, 139–147 (2022). https://doi.org/10.1016/j.tcs.2022.09.022. https://www.sciencedirect.com/science/article/pii/S0304397522005643

12. Tang, S.: Data summarization beyond monotonicity: non-monotone two-stage submodular maximization. In: International Conference on Combinatorial Optimization and Applications, pp. 277–286. Springer, Cham (2023)
13. Tang, S., Yuan, J.: Group equality in adaptive submodular maximization. arXiv preprint arXiv:2207.03364 (2022)
14. Thomas Kesselheim: Lecture notes (2021). https://tcs.cs.uni-bonn.de/lib/exe/fetch.php?media=teaching:ss21:vl-aau:lecture22.pdf

# (Independent) Roman Domination Parameterized by Distance to Cluster

Pradeesha Ashok[1], Gautam K. Das[2], Arti Pandey[3], Kaustav Paul[3](✉), and Subhabrata Paul[4]

[1] International Institute of Information Technology Bangalore, Bengaluru, India
pradeesha@iiitb.ac.in

[2] Department of Mathematics, Indian Institute of Technology Guwahati, Guwahati, India
gkd@iitg.ac.in

[3] Department of Mathematics, Indian Institute of Technology Ropar, Rupnagar, India
{arti,kaustav.20maz0010}@iitrpr.ac.in

[4] Department of Mathematics, Indian Institute of Technology Patna, Daulatpur, India
subhabrata@iitp.ac.in

**Abstract.** Given a graph $G = (V, E)$, a function $f : V \to \{0, 1, 2\}$ is said to be a *Roman Dominating function* (RDF) if for every $v \in V$ with $f(v) = 0$, there exists a vertex $u \in N(v)$ such that $f(u) = 2$. A Roman Dominating function $f$ is said to be an *Independent Roman Dominating function* (IRDF), if $V_1 \cup V_2$ forms an independent set, where $V_i = \{v \in V \mid f(v) = i\}$, for $i \in \{0, 1, 2\}$. The total weight of $f$ is equal to $\sum_{v \in V} f(v)$, and is denoted as $w(f)$. The *Roman Domination Number* (resp. *Independent Roman Domination Number*) of $G$, denoted by $\gamma_R(G)$ (resp. $i_R(G)$), is defined as $\min\{w(f) \mid f$ is an RDF (resp. IRDF) of $G\}$. For a given graph $G$, the problem of computing $\gamma_R(G)$ (resp. $i_R(G)$) is defined as the *Roman Domination problem* (resp. *Independent Roman Domination problem*).

In this paper, we examine structural parameterizations of the (Independent) Roman Domination problem. We propose fixed-parameter tractable (FPT) algorithms for the (Independent) Roman Domination problem in graphs that are $k$ vertices away from a cluster graph. These graphs have a set of $k$ vertices whose removal results in a cluster graph. We refer to $k$ as the distance to the cluster graph. Specifically, we prove the following results when parameterized by the deletion distance $k$ to cluster graphs: we can find the Roman Domination Number (and Independent Roman Domination Number) in time $4^k n^{O(1)}$. In terms of lower bounds, we show that the Roman Domination number can not be computed in time $2^{\epsilon k} n^{O(1)}$, for any $0 < \epsilon < 1$ unless a well-known conjecture, SETH fails. In addition, we also show that the Roman Domination problem, parameterized by distance to cluster, does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.

D. Du et al. (Eds.): COCOA 2024, LNCS 15435, pp. 157–169, 2025.
https://doi.org/10.1007/978-981-96-4448-3_13

**Keywords:** Roman Domination · Independent Roman Domination · FPT · Distance to Cluster · Kernel

## 1 Introduction

The concept of Roman Dominating function originated in an paper by Ian Stewart, titled "Defend the Roman Empire!" [12], published in Scientific American. Given a graph, where every vertex corresponds to a distinct geographical region within the historical narrative of the Roman Empire, the characterization of a location as secured or unsecured is delineated by the Roman Dominating function, denoted as $f$.

Specifically, a vertex $v$ is said to be unsecured if it lacks stationed legions, expressed as $f(v) = 0$. Conversely, a secured location is one where one or two legions are stationed, denoted by $f(v) \in \{1, 2\}$. The strategic methodology for securing an unsecured area involves the deployment of a legion from a neighboring location. In the fourth century A.D., Emperor Constantine the Great enacted an edict precluding the transfer of a legion from a fortified position to an unfortified one if such an action would leave the latter unsecured. Therefore, it is necessary to first have two legions at a given location ($f(v) = 2$) before sending one legion to a neighbouring location. Considering the substantial costs associated with legion deployment in specific areas, the Emperor aimed to strategically minimize the number of legions required to safeguard the Roman Empire.

Given a graph $G = (V, E)$, a *Roman Dominating function* (RDF) is defined as a function $f : V \to \{0, 1, 2\}$, where every vertex $v$, for which $f(v) = 0$ must be adjacent to at least one vertex $u$ with $f(u) = 2$. The weight of an RDF is defined as $w(f) = \sum_{v \in V} f(v)$. The *Roman Domination Number* is defined as $\gamma_R(G) = \min\{w(f) \mid f \text{ is an RDF of } G\}$. While the context is clear, if $f(v) = i$ for some RDF $f$, then we say that $v$ has label $i$.

Given a graph $G = (V, E)$, a set $S \subseteq V$ is defined as *independent set* if any two vertices of $S$ are non-adjacent. A function $f$ is referred to as an *Independent Roman Dominating function* (*IRDF*) if $f$ is an RDF and $V_1 \cup V_2$ is an independent set. The *Independent Roman Domination Number* is defined as $i_R(G) = \min\{w(f) \mid f \text{ is an IRDF of } G\}$. An IRDF $f$ of $G$ with $w(f) = i_R(G)$ is denoted as an $i_R(G)$-function of $G$. Given a graph $G = (V, E)$, the problem of computing $i_R(G)$ is known as *Independent Roman Domination problem.*

One of the objectives of parameterized complexity is to identify parameters that render NP-hard problems fixed-parameter tractable (FPT). This is of practical significance because there are often small parameters, aside from solution size, that capture important practical inputs. Hence, it only makes sense to explore problems under a multitude of parameters. There has recently been a lot of research in this area. A key research direction involves identifying a parameter as small as possible, under which a problem becomes fixed-parameter tractable or admits a polynomial-sized kernel. Structural parameterization involves a parameter that is a function of the input structure rather than the standard solution size. A continuing trend in structural parameterization is to study problems parameterized by the deletion distance to various graph classes where the problem is efficiently solvable.

Our parameter of interest is the 'distance' of the graph from a natural class of graphs. Here, 'distance' refers to the number of vertices that must be deleted from the graph to belong to the specified class. This paper focuses on one such special class of graphs: cluster graphs, where each connected component of the graph is a clique. Note that both Roman Domination and Independent Roman Domination problems can be easily solved in cluster graphs. Given a graph $G = (V, E)$, the minimum number of vertices that need to be deleted from the graph so that the remaining graph becomes a cluster graph is called *distance to cluster* of $G$ (denoted by CVD size of $G$).

## 2 Preliminaries

### 2.1 Graph Theoretic Notations

This paper only considers simple, undirected, finite and nontrivial graphs. Let $G = (V, E)$ be a graph. $n$ and $m$ will be used to denote the cardinalities of $V$ and $E$, respectively. $N(v)$ stands for the set of neighbors of a vertex $v$ in $V$ and $N[v] = N(v) \cup \{v\}$. For a set $S \subseteq V$, we define $N(S) = \cup_{v \in S} N(v)$. The number of neighbors of a vertex $v \in V$ defines its *degree*, which is represented by the symbol $deg(v)$. The maximum degree of the graph will be denoted by $\Delta$. For a set $U \subseteq V$, the notation $deg_U(v)$ is used to represent the number of neighbors that a vertex $v$ has within the subset $U$. Additionally, we use $N_U(v)$ to refer to the set of neighbors of vertex $v$ within $U$. Given a set $S \subseteq V$, $G \setminus S$ is defined as the graph induced on $V \setminus S$, that is $G[V \setminus S]$.

A vertex of degree one is known as a *pendant vertex.* A set $S \subseteq V$ is said to be a *dominating set* if every vertex of $V \setminus S$ is adjacent to some vertex of $S$. A graph $G$ is said to be a *complete graph* if any two vertices of $G$ are adjacent. A set $S \subseteq V$ is said to be a clique if the subgraph of $G$ induced on $S$ is a complete graph. A graph is said to be a *cluster graph* if every component of the graph is a clique. For every positive integer $n$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.

Given a graph $G = (V, E)$ and a function $f : V \to \{0, 1, 2\}$, $f_H : V(H) \to \{0, 1, 2\}$ is defined to be the function $f$ restricted on $H$, where $H$ is an induced subgraph of $G$.

For a graph $G = (V, E)$ and a function $f : V \to \{0, 1, 2\}$; we define $V_i = \{v \in V \mid f(v) = i\}$ for $i \in \{0, 1, 2\}$. The partition $(V_0, V_1, V_2)$ is said to be *ordered partition* of $V$ induced by $f$. Note that the function $f : V \to \{0, 1, 2\}$ and the ordered partition $(V_0, V_1, V_2)$ of $V$ have a one-to-one correspondence. So, when the context is clear, we write $f = (V_0, V_1, V_2)$. Given an RDF $f = (V_0, V_1, V_2)$, $(V_1, V_2)$ is said to be *Roman Dominating pair* corresponding to $f$. When the context is clear, we write Roman Dominating pair (omitting the notion of $f$).

## 2.2 Problem Definitions

Before presenting our results, we formalize the problems considered in the paper as follows:
SET-COVER

**Input**: An universe $U$ and a collection of subsets of $U$, $F = \{S_1, \ldots, S_m\}$ and a non-negative integer $k$.
**Question**: Does there exists $k$ sets $S_{i_1}, \ldots, S_{i_k}$ in $F$, such that $\bigcup_{j=1}^{k} S_{i_j} = U$?

RD-CVD

**Input**: A graph $G = (V, E)$, a cluster vertex deletion set $S$ and a non-negative integer $\ell$.
**Parameter**: $|S| = k$.
**Question**: Does there exists an RDF $f$ on $G$, with weight at most $\ell$?

RD-VC

**Input**: A graph $G = (V, E)$, a vertex cover $S$ and a non-negative integer $\ell$.
**Parameter**: $|S| = k$.
**Question**: Does there exists an RDF $f$ on $G$, with weight at most $\ell$?

IRD-CVD

**Input**: A graph $G = (V, E)$, a cluster vertex deletion set $S$ and a non-negative integer $\ell$.
**Parameter**: $|S| = k$.
**Question**: Does there exists an IRDF $f$ on $G$, with weight at most $\ell$?

$d$-Hitting Set

**Input**: An universe $U$ and a collection of subsets of $U$, $F = \{S_1, \ldots, S_m\}$ such that $|S_i| \leq d$, for all $i \in [m]$ and a non-negative integer $k$.
**Question**: Does there exists a subset $U' \subseteq U$, such that $|U'| \leq k$ and $U' \cap S_i \neq \emptyset$, for every $i \in [m]$?

## 2.3 Parameterized Complexity Notations and Definitions

*Conjecture 1.* [9] (Strong Exponential Time Hypothesis (SETH)) There is no $\epsilon > 0$ such that $\forall q \geq 3, q$-CNFSAT can be solved in $(2 - \epsilon)^n n^{\mathcal{O}(1)}$ time where $n$ is the number of variables in input formula.

**Theorem 1.** *[7] The SET-COVER problem can be solved in $2^n (m+n)^{\mathcal{O}(1)}$ time where $n$ is the size of the universe and $m$ is the size of the family of subsets of the universe.*

Due to space constraints, the other definitions and theorems are omitted from the manuscript.

### 2.4 Related Works

From the parameterized complexity point of view, it is surprising that there does not exist much literature on the Roman Domination problem (except [6,10]), while the classical dominating set problem is very well studied. Some related literature about the parameterized complexity of the domination problem can be found in [1,5,11]. One recent work about the domination problem parameterized by several structural parameters like distance to cluster and distance to split can be found in [8]. The techniques we designed in this paper are adaptations of the technique used in [8], with appropriate modifications to fit our problem.

In [6], Fernau proved that the Roman Domination parameterized by the solution size is $W[2]$-hard in general graphs, but FPT for planar graphs. He also showed that the same problem parameterized by treewidth is FPT. In [10], Mohannapriya et al. showed that a more generalized problem, that is $k$-Roman Domination problem parameterized by solution size is $W[1]$-hard, even for split graphs. To the best of our knowledge, no other parameterized complexity results exist for the Roman Domination problem.

### 2.5 Our Results

The main contribution of the paper is the following:

- In Sect. 3.1 (resp. Sect. 3.2), we show that the RD-CVD (resp. IRD-CVD) problem is FPT.
- In Sect. 4, we show that the RD-CVD problem cannot be solved in time $2^{\epsilon k} n^{O(1)}$ (where $0 < \epsilon < 1$) unless SETH (refer to Conjecture 1) fails, neither it admits a polynomial kernel unless NP $\subseteq$ coNP/poly.
- In Sect. 5, we conclude the paper with some future research directions.

## 3 Variants of Roman Domination Parameterized by CVD Size

In this section, we assume that a cluster vertex deletion set $S$ of size $k$ is given with the input graph $G = (V, E)$. If not, the algorithm mentioned in [2] can be used, which runs in $1.92^k n^{O(1)}$ time and outputs a cluster vertex deletion set of size at most $k$ or concludes that there does not any cluster vertex deletion set of size at most $k$.

### 3.1 Roman Domination

In this section, we propose an FPT algorithm for the Roman Domination problem when the parameter is CVD size.

We consider a CVD set $S$ as a part of the input, where $|S| = k$. Our algorithm starts with making a guess for $S_1 = V_1 \cap S$ and $S_2 = V_2 \cap S$, where $(V_1, V_2)$ is an optimal Roman Dominating pair. At first, a guess of $S_2$ is made from $S$. Then, the vertices of $N[S_2] \cap S$ are deleted from the graph. Then we guess $S_1$ from the remaining $S$ and delete $S_1$ from $S$.

Note that $S$ is a CVD set, hence $G \setminus S$ is disjoint union of cliques. Let $G \setminus S = C_1 \cup C_2 \cup \ldots C_q$, where every $C_i$ is a clique and $|C_i| = \ell_i$, for $i \in [q]$, Note that $q \leq n - k$. After the selection of $S_1$ and $S_2$, every clique belongs to exactly one of the following three types:

Type 0 ($T_0$) cliques: $C_i$ is a $T_0$ clique if every vertex of $C_i$ is adjacent to at least one vertex of $S_2$.

Type 1 ($T_1$) cliques: $C_i$ is a $T_1$ clique if exactly one vertex of $C_i$ is not adjacent to any vertex of $S_2$.

Type 2 ($T_2$) cliques: $C_i$ is a $T_2$ clique if $C_i$ contains at least two vertices which are not adjacent to any vertex of $S_2$.

We define an order $\rho_i$ on the vertices of the clique $C_i$ as follows: if $C_i$ is a $T_0$ or $T_2$ clique, then we order them arbitrarily; if $C_i$ is a $T_1$ clique, then the set $C_i \setminus N[S_2]$ contains exactly one vertex. We make that vertex the first vertex of $\rho_i$ and order the rest of the vertices of $C_i$ arbitrarily. Now we define an order $\rho$ on the vertex set of $G \setminus S$ as follows: $\rho = v_1, v_2, \ldots, v_{|G \setminus S|}$, where first $\ell_1$ vertices of $\rho$ are vertices of $C_1$ and follows the order $\rho_1$, then the next $\ell_2$ vertices of $\rho$ are vertices of $C_2$ and follows the order $\rho_2$ and so on. Now, here comes an observation.

**Observation 1.** *Given a $T_2$ clique $C_i$, any Roman Dominating pair $(V_1, V_2)$ extended from $(S_1, S_2)$ has one of the following properties:*

1. *$C_i \cap V_2 \neq \emptyset$.*
2. *A Roman Dominating pair $(V_1', V_2')$ can be extended from $(S_1, S_2)$, which has same or less weight than $(V_1, V_2)$ and $V_2' \cap C_i \neq \emptyset$*

The proof of this observation is omitted due to space constraints. From the above observation, we can rephrase the remaining problem as follows:

RD-DisjointCluster problem

**Input**: A graph $G = (V, E)$, a subset $S \subseteq V$ such that every component of $G \setminus S$ is a clique, a $(0,1,2)$-flag vector $f = (f_1, f_2, \ldots, f_q)$ corresponding to the cliques $(C_1, C_2, \ldots, C_q)$ and $\ell \in \mathbb{Z}^+$.
**Parameter**: $|S|$.
**Question**: Does there exists a subset $T \subseteq G \setminus S$ which satisfies all of the following conditions?
(a) For every $C_i$ with $f_i = 2$, $T \cap C_i \neq \emptyset$.
(b) $2|T| + g(T) \leq \ell$, where $g(T)$ = number of cliques with flag 1, which have empty intersection with $T$.

For an instance $(G, S, \ell)$ of the RD-CVD problem, with cliques $C_1, \ldots, C_q$ and the guesses of $S_1, S_2$, we build an instance $(\hat{G}, \hat{S}, f, \hat{\ell})$ of the RD-DisjointCluster problem as follows:

– $\hat{G} = G \setminus ((N[S_2] \cap S) \cup S_1)$.

- $\hat{S} = S \setminus ((N[S_2] \cap S) \cup S_1)$.
- $\hat{\ell} = \ell - 2|S_2| - |S_1|$.
- For all $i \in [q]$, $f_i = j$, if $C_i$ is a $T_j$ clique, $j \in \{0, 1, 2\}$.

It is not hard to observe that the instance $(G, S, \ell)$ of the RD-CVD problem and the instance $(\hat{G}, \hat{S}, f, \hat{\ell})$ of the RD-DisjointCluster problem are equivalent instances. This means $(G, S, \ell)$ is a YES instance if and only if $(\hat{G}, \hat{S}, f, \hat{\ell})$ is a YES instance.

**Formulation of the Problem as a Variant of Set Cover:** We define a variant of the set cover problem. Given an instance of the RD-DisjointCluster problem, we construct an instance of the set cover problem. Let $(G, S, f, \ell)$ be an instance of the RD-DisjointCluster problem and $\rho = (v_1, v_2, \ldots, v_{|G \setminus S|})$ be an ordering of the vertex set of $G \setminus S$, as defined earlier. We take the universe $U = S$ and $F = \{S_1, S_2, \ldots, S_{|G \setminus S|}\}$, where $S_i = N(v_i) \cap S$ for every $i \in [|G \setminus S|]$. Now, we modify the usual SET-COVER problem to suit our problem. The modified SET-COVER problem is defined below:

SET-CoverWithPartition problem (SCP)

**Input**: Universe $U$, a family of sets $F = \{S_1, S_2, \ldots, S_m\}$, a partition of $\beta = (\beta_1, \ldots, \beta_q)$ of $F$, a $(0, 1, 2)$ flag vector $f = (f_1, f_2, \ldots, f_q)$ corresponding to each block in the partition $\beta$ and a non-negative integer $\ell$.
**Parameter**: $|U|$.
**Question**: Does there exists a subset $F' \subseteq F$ which satisfies all of the following conditions?
(a) For every $\beta_i$ with $f_i = 2$, $F' \cap \beta_i \neq \emptyset$.
(b) $2|F'| + g(F') \leq \ell$, where $g(A)$ = number of blocks with flag 1, which have empty intersection with $A$, for $A \subseteq F$.

Given an instance $(G, S, f, \ell)$ of the RD-DisjointCluster problem, we define an instance $(U, F, \beta, f', \ell')$ of the SCP problem as follows:

- $U = S$.
- $F = \{S_i = N(v_i) \cap S \mid v_i \in G \setminus S\}$.
- $\beta_i = \{S_j \mid v_j \in C_i\}$, for $i \in [q]$.
- $f' = f$.
- $\ell' = \ell$.

It is not hard to show that the RD-DisjointCluster and SCP are equivalent problems.

**Observation 2.** *$(G, S, f, \ell)$ is a YES instance of the* RD-DisjointCluster *problem if and only if $(U, F, \beta, f', \ell')$ is a YES instance of the SCP problem.*

Now, we propose an algorithm to solve the SCP problem.

**Theorem 2.** *The* Set-CoverWithPartition *problem can be solved in $2^{|U|} O(m \cdot |U|)$ time.*

*Proof.* We propose a dynamic programming algorithm to solve the problem. For every $W \subseteq U$, $j \in [m]$ and $b \in \{0, 1, 2\}$, we define $OPT[W, j, b] := min_X\{2|X| + g_j(X)\}$, where $X$ satisfies the following properties:

1. $X \subseteq \{S_1, \ldots, S_j\}$.
2. $X$ covers $W$.
3. Let $\beta_x$ be the block that contains $S_j$. We redefine $f_x = b$, where $f_x$ is the flag associated with $\beta_x$. From every block $\beta_i$ $(i \leq x)$ with $f_i = 2$, at least one set from $\beta_i$ is in $X$.
4. The function $g_j$ is defined as follows. $g_j(X) :=$ number of blocks $\beta_i$ $(i \leq x)$ with $f_i = 1$, which have empty intersection with $X$.

Now, coming to the base case, for every $W \subseteq U$, with $W \neq \emptyset$ and $b \in \{0, 1, 2\}$; $OPT[W, 1, b] = 2$ if $W \subseteq S_1$, $OPT[W, 1, b] = \infty$, otherwise.

If $W = \emptyset$, $OPT[W, 1, b] = b$, for $b \in \{0, 1, 2\}$. To compute all the values of $OPT[W, j, b]$, we initially set all the remaining values to be $\infty$. We construct the following recursive formulation for $OPT[W, j + 1, b]$, for $j \geq 1$:

**Case 1:** $S_{j+1}$ is not the first set of the block $\beta_x$.

Note that two possibilities appear here. First, we pick $S_{j+1}$ in the solution $X$. Hence, we are left with the problem of covering $W \setminus S_{j+1}$ with some subset of $\{S_1, \ldots, S_j\}$ and since $S_{j+1}$ from the partition $\beta_x$ is already taken in solution, so the flag of $\beta_x$ can be reset to 0. Hence, in this case $OPT[W, j + 1, b] = 2 + OPT[W \setminus S_{j+1}, j, 0]$.

In the latter case, we do not pick $S_{j+1}$ in $X$; hence nothing is changed except the fact that now we need to cover $W$ with a subset of $\{S_1, \ldots, S_j\}$ and the flag of $\beta_x$ remains unchanged as $b$. Hence, $OPT[W, j + 1, b] = OPT[W, j, b]$.

So, $OPT[W, j + 1, b] = \min\{2 + OPT[W \setminus S_{j+1}, j, 0], OPT[W, j, b]\}$.

**Case 2:** $S_{j+1}$ is the first set of the block $\beta_x$. Here, three scenarios can appear:

**Case 2.1:** $b = 2$.
In this case, there is no option but to include $S_{j+1}$ in the solution as $b = 2$. Hence, we take $S_{j+1}$ in the solution and shift to the previous block. Now we need to cover $W \setminus S_{j+1}$ with a subset of $\{S_1, \ldots, S_j\}$. Hence, $OPT[W, j + 1, b] = 2 + OPT[W \setminus S_{j+1}, j, f_{x-1}]$.

**Case 2.2**: $b = 0$.
In this case, there are two choices, to include $S_{j+1}$ in the solution or not. If we include $S_{j+1}$ in the solution, then $OPT[W, j+1, b] = 2+OPT[W \setminus S_{j+1}, j, f_{x-1}]$. If we do not, then $OPT[W, j+1, b] = OPT[W, j, f_{x-1}]$. Hence $OPT[W, j+1, b] = \min\{2 + OPT[W \setminus S_{j+1}, j, f_{x-1}], OPT[W, j, f_{x-1}]\}$

**Case 2.3**: $b = 1$.
Similarly, in this case, there are two choices. If $S_{j+1}$ is included in the solution then $OPT[W, j+1, b] = 2+OPT[W \setminus S_{j+1}, j, f_{x-1}]$, by similar argument as above. If not, then $S_{j+1}$ has to contribute 1 in $OPT[W, j + 1, b]$, as at least one set from the block $\beta_x$ has to contribute 1 to $OPT[W, j + 1, b]$ and $S_{j+1}$ is the only set left

in $\beta_x$ at this moment. So, in this case, $OPT[W, j+1, b] = 1 + OPT[W, j, f_{x-1}]$. Hence, $OPT[W, j+1, b] = \min\{2 + OPT[W \setminus S_{j+1}, j, f_{x-1}], 1 + OPT[W, j, f_{x-1}]\}$.

We compute $OPT[W, j, b]$ in the increasing order of size of $W, j, b$. Hence, there are $3 \cdot 2^{|U|} \cdot m$ subproblems. It takes $|U|$ time to compute set differences (like $W \setminus S_{j+1}$). Hence, the time-complexity of our algorithm is $2^{|U|}O(m \cdot |U|)$. □

Hence, the following corollary can be concluded.

**Corollary 1.** *The* RD-DisjointCluster *problem can be solved in time* $2^{|S|}n^{O(1)}$.

**Theorem 3.** *The RD-CVD problem can be solved in time* $4^k n^{O(1)}$.

*Proof.* Given an instance $(G, S, \ell)$ of the RD-CVD problem and for every guess of $S_1', S_2' \subseteq S$ (with $|S_1'| = i_1$ and $|S_2'| = i_2$), we can construct an instance $(\hat{G}, \hat{S}, f, \hat{\ell})$ of the RD-DisjointCluster problem, which can be solved in time $2^{k-i_1-i_2}n^{O(1)}$. Hence, total time taken is $\sum_{i_1=1}^{k}(\binom{k}{i_1}\sum_{i_2=1}^{k-i_1}(\binom{k-i_1}{i_2}2^{k-i_1-i_2}))n^{O(1)} = 4^k n^{O(1)}$. □

In the next section, using a similar approach, we show that the IRD-CVD problem is also FPT.

## 3.2 Independent Roman Domination

In this section, we propose an FPT algorithm for the Independent Roman Domination problem when the parameter is the CVD size.

Similarly, like the case of Roman Domination, a guess for $S_1 = V_1 \cap S$ and $S_2 = V_2 \cap S$ is made, where $(V_1, V_2)$ is an optimal independent Roman Dominating pair. At first, we guess an independent set $S_2$ from $S$ and then delete all the vertices of $N[S_2]$ from the graph, as if our choice of $S_2$ is right, then all the vertices in $N(S_2) \setminus S_2$ should have label 0. Then, we choose another independent set $S_1$ from the remaining $S$ and delete all the vertices of $S_1$ and $N[S_1] \cap (G \setminus S)$ from the remaining graph. Note that if there exists a clique $C_i \subseteq N[S_1] \cup N[S_2]$ such that $C_i$ has a vertex $v$, that is not adjacent to any vertex of $S_2$, but it is adjacent to some vertex in $S_1$, then our choices of $S_1$, $S_2$ are incorrect, and we do not move further with these choices of $S_1$ and $S_2$.

Note that $S$ is a CVD set, hence $G \setminus S$ is disjoint union of cliques. Let $G \setminus S = C_1 \cup C_2 \cup \ldots C_q$, where every $C_i$ is a clique and $|C_i| = \ell_i$, for $i \in [q]$. Note that $q \leq n - k$. Note that, after the selection of $S_1$ and $S_2$ and the deletion process, every clique belongs to exactly one of the following two types:

Type 1 ($T_1$) cliques: $C_i$ is a $T_1$ clique if $C_i$ has exactly one vertex.

Type 2 ($T_2$) cliques: $C_i$ is a $T_2$ clique if $C_i$ has at least two vertices.

**Observation 3.** *Given a $T_2$ clique $C_i$, any independent Roman Dominating pair $(V_1, V_2)$ extended from $(S_1, S_2)$ has the following property: $C_i \cap V_2 \neq \emptyset$.*

*Proof.* $C_i$ is a $T_2$ clique, hence there exist at least two vertices $v_1, v_2$ in $C_i$. Note that both of them can not have non zero labels; at least one of them must have label 0. Without loss of generality, let $v_1$ have label 0, but $v_1$ does not have any neighbor in $S$, which has label 2, which implies a vertex in $C_i$ must have label 2. Hence, the result follows. $\square$

Hence the remaining problem can be rephrased as follows:

IRD-DisjointCluster problem

> **Input**: A graph $G = (V, E)$, a subset $S \subseteq V$ such that every component of $G \setminus S$ is a clique, a $(1, 2)$-flag vector $f = (f_1, f_2, \ldots, f_q)$ corresponding to the cliques $(C_1, C_2, \ldots, C_q)$ and $\ell \in \mathbb{Z}^+$.
> **Parameter**: $|S|$.
> **Question**: Does there exists a subset $T \subseteq G \setminus S$ which satisfies all of the following conditions?
> (a) For every $C_i$ with $f_i = 2$, $|T \cap C_i| = 1$.
> (b) $2|T| + g(T) \leq \ell$, where $g(T) =$ number of cliques with flag 1, which have empty intersection with $T$.

For an instance $(G, S, \ell)$ of the IRD-CVD problem, with cliques $C_1, \ldots, C_q$ and the guesses of $S_1, S_2$, we build an instance $(\hat{G}, \hat{S}, f, \hat{\ell})$ of the IRD-DisjointCluster problem as follows:

- $\hat{G} = G \setminus (S_1 \cup N[S_2] \cup (N(S_1) \cap (G \setminus S))$.
- $\hat{S} = S \setminus (N[S_2] \cup S_1)$.
- $\hat{\ell} = \ell - 2|S_2| - |S_1|$.
- $f_i = j$, if $C_i$ is a $T_j$ clique, $i \in [q]$ and $j \in \{1, 2\}$.

**Formulation of the problem as a variant of set cover**: We define a variant of the set cover problem similarly to the Roman Domination problem. Given an instance of the IRD-DisjointCluster problem, we construct an instance of the set cover problem. Let $(G, S, \ell, f)$ be an instance of the IRD-DisjointCluster problem and $\rho$ be any arbitrary order of the vertex set $G \setminus S$. We take the universe $U = S$ and $F = \{S_1, S_2, \ldots, S_{|G \setminus S|}\}$, where $S_i = N(v_i) \cap S$ for every $i \in [|G \setminus S|]$. Now, we modify the usual set cover problem to suit our problem. The modified set cover problem is defined below:

Independent-Set-CoverWithPartition problem (ISCP)

> **Input**: Universe $U$, a family of sets $F = \{S_1, S_2, \ldots, S_m\}$, a partition of $\beta = (\beta_1, \ldots, \beta_q)$ of $F$, a $(1, 2)$ flag vector $f = (f_1, f_2, \ldots, f_q)$ corresponding to each block in the partition $\beta$ and a non-negative integer $\ell$.
> **Parameter**: $|U|$.
> **Question**: Does there exists a subset $F' \subseteq F$ which satisfies all of the following conditions?
> (a) For every $\beta_i$ with $f_i = 2$, $|F' \cap \beta_i| = 1$.
> (b) $2|F'| + g(F') \leq \ell$, where $g(A) =$ number of blocks with flag 1, which have empty intersection with $A$; for $A \subseteq F$.

Given an instance $(G, S, f, \ell)$ of the IRD-DisjointCluster problem, we define an instance $(U, F, \beta, f', \ell')$ of the ISCP problem as follows:

- $U = S$.
- $F = \{S_i = N(v_i) \cap S \mid v_i \in G \setminus S\}$.
- $\beta_i = \{S_j \mid v_j \in C_i\}$, for $i \in [q]$.
- $f' = f$.
- $\ell' = \ell$.

**Observation 4.** *$(G, S, f, \ell)$ is a YES instance of the* IRD-DisjointCluster *problem if and only if $(U, F, \beta, f', \ell')$ is a YES instance of the ISCP problem.*

Next, we propose an algorithm to solve the ISCP problem.

**Theorem 4.** *The* Independent-Set-CoverWithPartition *problem can be solved in $2^{|U|}O(m \cdot |U|)$ time.*

*Proof.* The proof is omitted due to space constraints. □

Hence, the following corollary can be concluded.

**Corollary 2.** *The* IRD-DisjointCluster *problem can be solved in time $2^{|S|}n^{O(1)}$.*

**Theorem 5.** *The IRD-CVD problem can be solved in time $4^k n^{O(1)}$.*

*Proof.* The proof is analogous to the proof of Theorem 3. □

## 4 Lower Bounds

In this section, we propose a lower bound on the time-complexity of the RD-CVD problem. We also show that the RD-CVD (resp. RD-VC) problem does not admit a polynomial kernel (recall that RD-VC is the Roman Domination problem parameterized by vertex cover number).

First, we provide the lower bound for the RD-CVD problem. Below, we state a necessary result from the existing literature (refer to Theorem 1.1 in [3]).

**Theorem 6.** *[3] The following statement is equivalent to SETH: For every $\epsilon < 1$, there exists $d \in \mathbb{Z}^+$, such that the $d$-*Hitting Set *problem for set systems over $[n]$ can not be solved in time $O(2^{\epsilon n})$.*

Now, we show a reduction from the $d$-Hitting Set problem to the RD-CVD problem to show a similar lower bound like Theorem 6 for the RD-CVD problem.

**Theorem 7.** *There exists a polynomial time algorithm that takes an instance $(U, F, t)$ of the $d$-*Hitting Set *problem and outputs an instance $(G, 2t)$ of the RD-CVD problem (and the RD-VC problem), where $G$ has a cluster vertex deletion set (and vertex cover) of size $|U|$; and $(U, F)$ has a $d$-*Hitting Set *of size at most $t$ if and only if $G$ has a Roman Dominating function of size at most $2t$.*

*Proof.* The proof is omitted due to space constraints. □

Hence, by Theorem 6 and 7, we can prove the following theorem.

**Theorem 8.** *The RD-CVD (and RD-VC) problem can not be solved in time* $2^{\epsilon k} n^{O(1)}$*, for any* $0 < \epsilon < 1$*, unless SETH fails.*

*Proof.* Let the RD-CVD (or RD-VC) problem be solved in time $2^{\epsilon k} n^{O(1)}$. Then by Theorem 7, we can solve the $d$-Hitting Set problem with $|U| = k$ in $2^{\epsilon k} n^{O(1)}$ time. Hence, by Theorem 6, this contradicts the SETH. Hence, the RD-CVD (and RD-VC) problem can not be solved in time $2^{\epsilon k} n^{O(1)}$, unless the SETH fails. □

We state another theorem to show that the RD-CVD (and RD-VC) problem is unlikely to admit a polynomial kernel.

**Theorem 9.** *[4] The $d$-*Hitting Set *problem parameterized by the universe size does not admit any polynomial kernel unless NP* $\subseteq$ *coNP/poly.*

Hence combining the above discussion with Theorem 9, the following theorem can be concluded.

**Theorem 10.** *The RD-CVD (and RD-VC) problem does not admit a polynomial kernel unless NP* $\subseteq$ *coNP/poly.*

*Proof.* By Theorem 9, the $d$-Hitting Set problem parameterized by universe size does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. Since the reduction provided in Theorem 7 is a polynomial parameter transformation (PPT), the RD-CVD and RD-VC do not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. □

## 5 Conclusion

In this work, we have extended the study on the parameterized complexity of the Roman Domination problem and one of its variants. There are other interesting structural parameters, such as neighborhood diversity and cliquewidth, for which it would be interesting to determine whether the problem parameterized by these parameters is fixed parameter tractable (FPT).

Another promising research direction is to develop an algorithm to solve the RD-CVD problem with better time-complexity than $4^k n^{O(1)}$. Given that the lower bound on time-complexity mentioned in Theorem 8, it might be possible to achieve an improved algorithm.

**Acknowledgement.** Research of Arti Pandey is supported by CRG project, Grant Number-CRG/2022/008333, Science and Engineering Research Board (SERB), India.

## References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating set. J. ACM **51**(3), 363–384 (2004)
2. Boral, A., Cygan, M., Kociumaka, T., Pilipczuk, M.: A fast branching algorithm for cluster vertex deletion. Theory Comput. Syst. **58**(2), 357–376 (2016)
3. Cygan, M., et al.: On problems as hard as CNF-SAT. ACM Trans. Algorithms (2016)
4. Dom, M., Lokshtanov, D., Saurabh, S.: Kernelization lower bounds through colors and ids. ACM Trans. Algorithms **11** (2014)
5. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: on completeness for W. Theor. Comput. Sci. **141**(1&2), 109–131 (1995)
6. Fernau, H.: Roman domination: a parameterized perspective. Int. J. Comput. Math. **85**(1), 25–38 (2008)
7. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Texts in Theoretical Computer Science. An EATCS Series. Springer, Cham (2010)
8. Goyal, D., Jacob, A., Kumar, K., Majumdar, D., Raman, V.: Parameterized complexity of dominating set variants in almost cluster and split graphs. CoRR, abs/2405.10556 (2024)
9. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
10. Mohanapriya, A., Renjith, P., Sadagopan, N.: Roman k-domination: hardness, approximation and parameterized results. In Lin, C.-C., Lin, B.M.T., Liotta, G. (eds.) WALCOM: Algorithms and Computation - 17th International Conference and Workshops, WALCOM 2023, Hsinchu, Taiwan, 22–24 March 2023, Proceedings. Lecture Notes in Computer Science, vol. 13973, pp. 343–355. Springer, Cham (2023)
11. Philip, G., Raman, V., Sikdar, S.: Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. ACM Trans. Algorithms **9**(1), 11:1–11:23 (2012)
12. Stewart, I.: Defend the roman empire! Sci. Am. **281**, 136–138 (1999)

# Fair Selection of Clearing Schemes for Kidney Exchange Markets

Robert D. Barish(✉) and Tetsuo Shibuya

Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan
rbarish@ims.u-tokyo.ac.jp, tshibuya@hgc.jp

**Abstract.** For the Kidney Exchange Problem (KEP), one has a barter exchange market represented by a digraph with vertices corresponding to either immunologically incompatible donor-acceptor pairs, non-directed donors, cadavers, or unpaired recipients, and directed edges corresponding to possible kidney exchanges. The objective is then to solve the associated clearing problem of finding an above threshold weight partition of the network into vertex-disjoint transplant cycles and/or paths. In this work – with a primary motivation being the broad applicability of the KEP model to barter exchange markets of indivisible goods – we conduct a theoretical investigation of the problem of uniformly, and in this sense "fairly", sampling witnesses for a formalization of the KEP we denote KEP-$(L_c, L_p, \Upsilon)$, where we have cycle and path vertex-wise length constraints $L_c$ and $L_p$, respectively, and where we require that the sum of all edge weights in a partition is at least $\Upsilon \in \mathbb{N}_0$. Here, for KEP-$(\infty, \infty, 0)$, we provide an $\mathcal{O}\left(4^g \cdot n^4 \cdot m\right)$ time uniform sampling scheme (assuming access to an idealized coin flipping oracle) for networks on $n$ vertices and $m$ edges admitting bimodal embeddings (i.e., embeddings where each set of edges oriented away from a given vertex occur contiguously in a rotational ordering of edges incident to the vertex) on genus $\leq g$ surfaces, as well as a Fully Polynomial-time Almost Uniform Sampling (FPAUS) scheme for arbitrary genus digraphs. Subsequently, taking inspiration from recent rapid experimental advances in using boson sampling (respectively, Guassian boson sampling) to approximate the permanents (respectively, hafnians) of complex matrices, we reduce the uniform sampling problem for KEP-$(L_c, L_p, \Upsilon)$ to calculating permanents of hollow Hermitian $\{-1, 0, 1\}$ matrices. However, we also moderate this latter finding by showing that no Fully Polynomial-time Randomized Approximation Scheme (FPRAS) for the permanent of such matrices unless $NP = RP$.

**Keywords:** kidney exchange problem · clearing problem · barter exchange market · algorithmic fairness · matrix permanent · FPAUS · FPRAS

This work was supported by JSPS Kakenhi grants {20K21827, 20H05967, 21H04871}, and JST CREST Grant JPMJCR1402JST.

D. Du et al. (Eds.): COCOA 2024, LNCS 15435, pp. 170–185, 2025.
https://doi.org/10.1007/978-981-96-4448-3_14

## 1 Introduction

Shortages of critical resources coupled with trust deficits and liquidity constraints (whether actual or imposed by legal and ethical barriers), can necessitate a return to the barter system from antiquity, wherein non-monetary goods and services are pairwise exchanged between two or more parties. This was most recently evidenced by the swapping of scarce medical supplies between hospitals during the COVID-19 pandemic [5], as well as the reciprocal exchange of vaccines between nations [65]. The economy for human organs (e.g., kidneys, hearts, livers, and corneas) is a well-known limit case of this phenomena, as there is an extreme excess of demand relative to supply, no contract can reasonably compel the later donation of an organ in return for one received, and the laws of nearly every nation on earth (with the exception of Iran [28]) prohibit monetary compensation in exchange for human organs. This supply-demand discrepancy has been most pronounced for human kidneys, where in 2020 in the United States alone, $91,099$ patients were on a waiting list to receive a kidney and only $22,817$ kidney transplants were performed [35].

While the majority of transplanted kidneys are derived from cadavers, the lack of suitable kidneys and the relative safety of renal transplantation has given rise to *kidney exchange schemes* with living donors. Here, these schemes correspond to barter exchange markets consisting of multiple (typically) blood-type (ABO) or Human Leukocyte Antigen (HLA) incompatible donor-acceptor pairs who cooperate to form cycles, and provided the existence of an appropriate live or cadaver donor, chains of feasible kidney donations [3,4,9,10,20,42,49–53]. This has, in turn, led to significant interest in algorithms and other heuristics for solving the clearing problem for kidney exchanges.

To formalize these notions, we can consider the following graph theoretic interpretation of barter exchange markets:

**Definition 1:** *Barter exchange market.* We say that a barter exchange market corresponds to a digraph $G$, in which vertices represent agents with at most a single instance of an indivisible good, where directed edges correspond to feasible transfers of goods between agents, and where weights on directed edges can be used to specify the utility increase for the beneficiary of a transfer.

We can next formalize the *Kidney Exchange Problem* (KEP) as follows:

**Problem:** *Kidney Exchange Problem with Cycle Length Constraint $L_c$, Path Length Constraint $L_p$, and minimum weight requirement $\Upsilon$; KEP-$(L_c, L_p, \Upsilon)$*
**Input:** A barter exchange market corresponding to a digraph $G$ with vertex set $V_G = V_X \cup V_Y \cup V_Z$, edge set $E_G$, and a set $W_G$ of integer weights, where $V_X \subseteq V_G$ corresponds to immunologically incompatible donor-acceptor pairs, $V_Y \subset V_G$ corresponds to live or cadaver donors, $V_Z \subset V_G$ corresponds to pure acceptors, $E_G$ corresponds to possible kidney exchanges between compatible donors and recipients, and each edge $e_i \in E$ has a corresponding positive integer weight $w_i \in W_G$; parameters $L_c, L_p, \Upsilon \in \mathbb{N}_0$.
**Objective:** Find a set of edges that partitions $G$ by inducing a set of directed

cycles and paths – where cycles and paths have at most $L_c$ and $L_p$ vertices, respectively – spanning all vertices in $V_G$, and where a sum over the edge weights in the partition is at least $\Upsilon$.

For an illustrative example, in Fig. 1(a) we show a barter exchange market that could serve as input for KEP-$(L_c, L_p, \Upsilon)$, where half-filled nodes correspond to incompatible donor-acceptor pairs and highlighted edges indicate a possible witness for the clearing problem. In Fig. 1(b) we show a variation on this example with a non-directed donor (white node) and pure acceptor (black node). In both cases, edges can be weighted by any number of criterion, ranging from ABO and HLA compatibility issues to quantized value judgments [22,30].

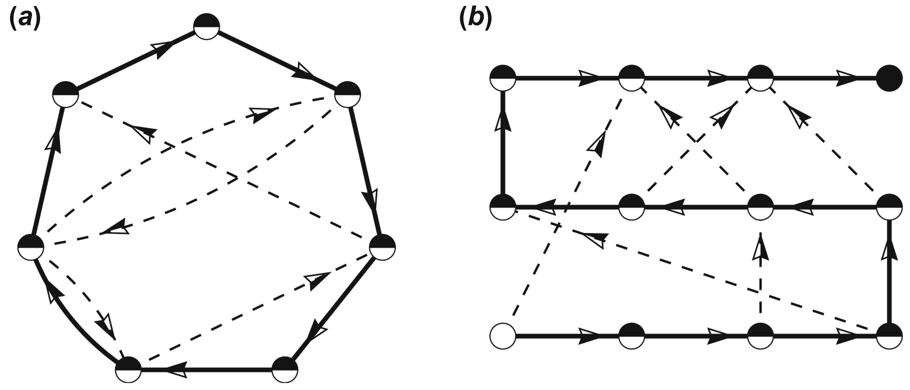


**Fig. 1.** Illustrations of example barter exchange markets; **(a)** example barter exchange market where half-filled nodes correspond to incompatible donor-acceptor pairs, weighted directed edges correspond to possible exchanges between these pairs, and highlighted edges show an example solution to the clearing problem; **(b)** variation on (a) with a non-directed donor (white node) and pure acceptor (black node).

In this work, we conduct a theoretical investigation of the problem of fairly choosing clearing schemes for the notion of barter exchange markets from Definition 1 – with "fairness" defined in roughly the same sense as [26,27], where the authors rely on Integer Linear Programming (ILP) and Constraint Programming (CP) brute force techniques – by sampling as uniformly as possible from maximum weight or above threshold weight sets of witnesses for KEP-$(L_c, L_p, \Upsilon)$. Consider that the use of an algorithm to choose a particular clearing scheme, potentially among multiple optimal or near optimal solutions, can lead to certain individuals or groups having a higher or lower chance of receiving a kidney, or perhaps even their exclusion from the exchange. Accordingly, common sense dictates that such a decision must not be subject to the particularities of an algorithm's implementation (e.g., where the order in which network edges are specified has bearing on the choice of a witness), or other sources of incidental bias among healthcare practitioners. We can also refer the reader to [21,45,57] for other fairness considerations, and to [24] for a differential-privacy-like formalization of individual fairness.

However, in conducting this analysis, our objective is also to look beyond the practical restrictions for the KEP, or more specifically, the requirement that transplant cycles and chains be as small as possible (e.g., involving $\leq 3$ individuals [3]) to minimize the complexity and risks of simultaneous surgeries. This is primarily because the KEP serves as a simplified model for barter exchange markets of indivisible goods in scenarios where much longer cycles and chains of exchanges are permissible. Consider, for example, that the KEP-$(L_c, L_p, \Upsilon)$ problem applies equally well to situations where the agents in our barter exchange are amputees or people with different sized feet looking to satisfy coincidences of wants by exchanging shoes [15,47]. Likewise, we can consider the applicability of the KEP-$(L_c, L_p, \Upsilon)$ model to exchanges where agents pairwise swap books to save on costs or perhaps access out-of-print materials [12,31].

We begin by considering the KEP-$(\infty, \infty, \Upsilon)$ problem, and by introducing the notion of *In-component Out-component split (IO-split) genus* for digraphs:

**Definition 2:** *In-component Out-component split (IO-split) genus.* We say that a digraph has IO-split genus $g$ if and only if each of its vertices $v_i$ can be replaced with a pair of vertices, $v_{(i,a)}$ and $v_{(i,b)}$, such that reattaching all edges formerly oriented towards $v_i$ (resp. oriented away from $v_i$) to $v_{(i,a)}$ (resp. $v_{(i,b)}$) yields a genus $g$ digraph.

In Sect. 3, letting $G$ be an IO-split genus $g$ digraph with $n$ vertices and $m$ edges, and assuming access to an idealized coin flipping oracle, we first show that an $\mathcal{O}\left(4^g \cdot n^4 \cdot m\right)$ time (or $\mathcal{O}\left(n^4 \cdot m\right)$ time if $g = 0$) uniform random sampler exists for the witness set of KEP-$(\infty, \infty, 0)$ (Theorem 1 and Corollary 1). Next, in the case where $g$ is unbounded but edge weights are restricted to be non-negative integers, we show that an $\mathcal{O}\left(\Psi \cdot n \cdot m\right)$ time Fully Polynomial-time Almost Uniform Sampler (FPAUS) exists, where $\Psi \approx \mathcal{O}\left(n^7 \cdot \ln^4(n)\right)$ [46] is the time complexity of the Jerrum-Sinclair-Vigoda Fully Polynomial-time Randomized Approximation Scheme (FPRAS) [37] for the non-negative matrix permanent (Corollary 2). With regard to the search problem for KEP-$(\infty, \infty, \Upsilon)$, we also show that a witness can be found in $\mathcal{O}\left(n^{\frac{1}{2}} \cdot m \cdot \ln(n \cdot C)\right)$ and $\mathcal{O}\left(n^{\frac{4}{3}} \cdot \ln^2(n) \cdot \ln(n \cdot C)\right)$ time for arbitrary and IO-split genus 0 barter exchange markets, respectively, where $C$ is the maximum absolute value of an edge weight (Corollary 3).

In the proceeding Sect. 4, we detail a method of constructing a uniform random sampler for general KEP-$(L_c, L_p, \Upsilon)$ instances provided a set of pre-enumerated cycles and paths, though one relying on a computationally difficult subroutine of computing a hollow Hermitian $\{-1, 0, 1\}$ matrix permanent (Theorem 2). We then proceed to moderate this result by proving that, unless $NP = RP$, no FPRAS can exist for approximating permanents of such matrices (Theorem 3).

Despite the challenges inherent in evaluating $\{-1, 0, 1\}$ matrix permanents, a key motivation for Sect. 4 is the substantial amount of work over the past decade on both the theory and practical implementation of non-adaptive quantum computing schemes (i.e., quantum computing without incremental adjustments or fine-tuning using previous results) based on boson sampling [1,2,43] and Gaussian boson sampling [34]. This has included surprisingly impressive experimental demonstrations [7,13,19,44,56,63,64,66], including some directly concerning the use of boson sampling and related schemes to solve graph theoretic problems [6,19]. We remark that Bradler et al. [13] has also provided an explicit method of using Gaussian boson sampling to approximate the number of perfect matchings in a graph. Here, as the means of demonstrating quantum supremacy via either boson sampling or Gaussian boson sampling fundamentally concerns faster-than-classically-possible approximation of the permanent (in the case of boson sampling) and hafnian (in the case of Gaussian boson sampling) for complex matrices, either method provides a direct means of approximating the matrix permanents that arise in the context of the Theorem 2 sampler. In this context, we can also pose the question as to how much of a speedup can be obtained for approximating the non-negative matrix permanent via quantum methods – in particular, via boson sampling or Gaussian boson sampling – relative, say, to using the Jerrum-Sinclair-Vigoda FPRAS [37].

## 2 Clarifications and Preliminaries

### 2.1 Graph Theoretic Concepts and Terminology

We will generally follow terminology from Diestel [23], or where appropriate, Bondy and Murty [11]. All graphs and digraphs in this work should be assumed to be simple (i.e., free of multiedges, loops, and parallel directed edges), though antiparallel directed edges between the same pair of vertices is allowed. To briefly cover some less common terminology, we say that a digraph has *bimodal* genus $g$ – or when $g = 0$, that a digraph is *bimodal planar* – if it is embeddable on a genus $g$ surface without edge crossings, vertex-edge crossings, or vertex-edge overlaps, and where all edges oriented away from each vertex occur contiguously in a rotational ordering of the edges incident to the vertex. Also, a *partition* of a graph or digraph $G$ into a specified set of graphs $s_1, s_2, \ldots, s_n \in S$ corresponds to a set of vertex disjoint subgraphs of $G$, each isomorphic to an instance of a graph in $S$, covering all vertices of $G$.

### 2.2 Complexity of Approximate Counting

Letting $\phi$ be an arbitrary instance of a counting or optimization problem $\#X$, letting $f(\phi)$ be a function which returns an optimal solution for $\phi$, and letting $\widehat{f}(\phi)$ be a *Polynomial Time Approximation Scheme* (PTAS) for $\#X$, it must hold that $(1-\epsilon) f(\phi) \leq \widehat{f}(\phi) \leq (1+\epsilon) f(\phi)$, where $\widehat{f}(\phi)$ must also run in time polynomial in $|\phi|$ for every fixed *error parameter* $\epsilon > 0$. A *Randomized Approximation Scheme* (RAS) is defined similarly, where letting $\phi$ and $\#X$ be defined as before, and letting $\widehat{f}(\phi)$ be a RAS for $\#X$, we have that

*Prob* $\left[e^{-\epsilon} \leq \left(\frac{\widehat{f}(\phi)}{f(\phi)}\right) \leq e^{\epsilon}\right] \geq 1 - \delta$ for some *error parameter* $\epsilon > 0$ and *accuracy parameter* $0 < \delta < 1$ (where we typically set $\delta = \frac{1}{4}$). We call the aforementioned RAS a *Polynomial-time Randomized Approximation Scheme* (PRAS) if its runtime is polynomial in $|x|$ (but not necessarily $\epsilon$). If a PTAS (resp. PRAS) runs in time polynomial in $|\phi|$ and $\epsilon^{-1}$, then we refer to the algorithm as a *Fully Polynomial Time Approximation Scheme* (FPTAS) (resp. *Fully Polynomial-time Randomized Approximation Scheme* (FPRAS)).

In this work we will make use of what are known as *Approximating Preserving (AP) reductions*, which are a powerful tool for establishing the existence or non-existence of a FPRAS for a given counting or optimization problem. Following Dyer et al. [25], an AP reduction from an integer counting problem $\#Y$ to an integer counting problem $\#X$ consists of a probabilistic oracle Turing machine $M$ satisfying the following three conditions: (condition 1) all inputs to $M$ are of the form $(x, \epsilon)$, where $x$ is an instance of $\#X$ and $0 < \epsilon < 1$ is an error parameter; (condition 2) $M$ is a RAS for $\#Y$ whenever the oracle is a RAS for $\#X$; and (condition 3) $M$ runs in time polynomial in both $|x|$ and $\epsilon^{-1}$. Here, if $\#Y$ is AP-reducible to $\#X$ and vice versa, then we call $\#X$ and $\#Y$ *AP-interreducible*, and write $\#X \equiv_{AP} \#Y$. We can now note that if a FPRAS exists for an integer counting problem $\#X$, and if $\#Y$ is AP-reducible to $\#X$, then a FPRAS likewise exists for $\#Y$. On the other hand, if $\#SAT$ or any problem AP-interreducible with $\#SAT$ (note that $\#SAT$ is trivially polynomial time reducible to any problem in $\#P$) is AP-reducible to a given problem $\#Y$, then it is known that $\#Y$ cannot admit a FPRAS unless $NP = RP$ [25].

### 2.3 Uniform and Almost Uniform Sampling

Following [36], we define the *total variation distance* between a pair of distributions $\mathcal{P}$ and $\mathcal{Q}$ on a countable set $\Omega$ as $||\mathcal{P} - \mathcal{Q}||_{TV} := \frac{1}{2}\sum_{x\in\Omega} |\mathcal{P}(x) - \mathcal{Q}(x)| = \max_{A\subseteq\Omega} |\mathcal{P}(A) - \mathcal{Q}(A)|$. Here, we call a randomised algorithm a *almost uniform sampler* if it accepts a string $x \in \Sigma^*$ and a *sampling tolerance* $\delta > 0$, then outputs witnesses $w_1, w_2, \ldots \in W \subseteq S(x)$ from a solution set $S(x)$, such that the total variation distance between the distribution for $W$ and a uniform distribution on $S(x)$ is $\leq \delta^3$. We call such an almost uniform sampler a *Fully Polynomial-time Almost Uniform Sampler* (FPAUS) if its run time is a polynomial function of $|x|$ and $\ln\left(\delta^{-1}\right)$. In this work (e.g., in Theorem 1), we will also refer to a *uniform sampler* defined in the same manner as an almost uniform sampler, with the exception that it is able to make constant time calls to an idealized coin flipping oracle, thus allowing the total variation distance between the distribution for $W$ and a uniform distribution on $S(x)$ to be equal to zero.

## 3 Uniform and Almost Uniform Sampling Schemes for KEP-$(\infty, \infty, 0)$ Witnesses

**Theorem 1.** *Letting $G$ be an IO-split genus $0$ digraph with $n$ vertices and $m$ edges, and assuming access to an idealized coin flipping oracle, an $\mathcal{O}\left(n^4 \cdot m\right)$ time uniform sampler exists for KEP-$(\infty, \infty, 0)$ witnesses.*

*Proof.* Let $G$ be an edge-weighted bimodal planar or IO-splittable genus 0 digraph serving as input for an instance of KEP-$(\infty, \infty, \Upsilon)$, having vertex set $V_G$ where $n = |V_G|$, edge set $E_G$ where $m = |E_G|$, and a set of integer edge weights $w_1, w_2, \ldots \in W$ where weight $w_i \in W$ corresponds to edge $e_i \in E_G$. Let $V_G = V_X \cup V_Y \cup V_Z$, where $V_X$ corresponds to immunologically incompatible donor-acceptor pairs, $V_Y$ corresponds to live or cadaver donors, and $V_Z$ corresponds to pure acceptors. We will proceed by first detailing a folklore result that partitioning $G$ into a set of directed cycles and/or directed paths, where the sum of all edge weights is $\geq \Upsilon$, can be reduced in linear time to the problem of finding a 1-factor of a bipartite undirected graph $H$ where the sum of all edge weights in the 1-factor is $\geq \Upsilon$ (see, e.g., [3] for previous application of this observation to kidney exchanges).

To begin, we construct a graph $H$ from $G$ via the following four steps: (step 1) for each vertex $v_i \in V_X$ we generate a pair of vertices $v_{(i,in)}$ and $v_{(i,out)}$; (step 2) for every vertex $v_i \in V_Y$ we generate a vertex $v_{(i,out)}$ in $H$; (step 3) for every vertex $v_i \in V_Z$ we generate a vertex $v_{(i,in)}$ in $H$; (step 4) for every edge $e_i \in E_G$ with weight $w_i \in W$, where $e_i$ corresponds to $v_a \rightarrow v_b$ for some $v_a, v_b \in V_G$, we add an edge $v_{(a,out)} \leftrightarrow v_{(b,in)}$ of weight $w_i$ to $H$. Here, we can observe that $H$ will be a bipartite graph regardless of whether $G$ is bipartite, and moreover, a planar graph if $G$ is an IO-split genus 0 digraph.

To now see that 1-factors of $H$ are in bijection with witnesses for KEP-$(\infty, \infty, \Upsilon)$ with $G$ as input, we make the following observations: (obs. 1) edges in a 1-factor of $H$ will be the form $v_{(i,out)} \leftrightarrow v_{(j,in)}$; (obs. 2) edges of the form $v_{(i,out)} \leftrightarrow v_{(j,in)}$ have a direct correspondence to (and the same weight as) directed edges in $G$ of the form $v_i \rightarrow v_j$; (obs. 3) identifying the pairs of vertices generated in (step 1) will cause any 1-factor of $H$ to become a set of cycles and/or paths, where one endpoint of any path will be a vertex generated in (step 2), corresponding to a vertex $v_i \in V_Y$, and one endpoint of any path will be a vertex generated in (step 3), corresponding to a vertex $v_i \in V_Z$. Accordingly, we have that identifying the pairs of vertices generated in (step 1) will transform a 1-factor of $H$ into a set of cycles and/or paths having a direct correspondence to (and the same total weight as) a set of directed cycles and/or directed paths in $G$ serving as a witness for KEP-$(\infty, \infty, \Upsilon)$.

We next observe that the search problem of finding a perfect matching for $H$ is well-known to be self-reducible in the strong sense of Schnorr [54,55], allowing us to choose a perfect matching for $H$ uniformly at random provided access to an idealized coin flipping oracle. To do so we simply select an edge in $H$ with a probability proportional to the number of perfect matchings that it participates in, delete the edge as well as its endpoints, and recurse the process until the set of deleted edges constitutes a perfect matching. Here, if $H$ is planar, we can exactly count the number of unweighted perfect matchings in $H$ using the $\mathcal{O}\left(n^3\right)$ Fisher-Kasteleyn-Temperley (FKT) algorithm [38–40,58], can moreover determine the number of perfect matchings each edge participates in using $m \in \mathcal{O}(n)$ calls to the FKT algorithm, and once again provided access to an idealized coin flipping oracle, can construct a uniform sampler for unweighted perfect matchings via the aforementioned recursive procedure. Putting everything together, as we specify $\Upsilon = 0$ and can therefore treat all edge weights in $H$ as being equal to 1, this yields a uniform sampler for KEP-$(\infty, \infty, 0)$ witnesses having the stated time complexity of $\mathcal{O}\left(n^4 \cdot m\right)$.

The proof argument for Theorem 1 now yields the following corollaries:

**Corollary 1.** *Letting $G$ be an IO-split genus $g$ digraph with $n$ vertices and $m$ edges, and assuming access to an idealized coin flipping oracle, an $\mathcal{O}\left(4^g \cdot n^4 \cdot m\right)$ time uniform sampler exists for KEP-$(\infty, \infty, 0)$ witnesses.*

*Proof.* Observe that the number of perfect matchings in a graph of genus $g$ can be expressed as a linear combination of $4^g$ Pfaffians [33,48,59], or alternatively, determined by running the FKT algorithm [38–40,58] on $4^g$ instances of planar graphs [18]. It now suffices to observe that the pre-processing steps in each of these approaches will not affect the asymptotic time complexity for the uniform sampler from Theorem 1, and that we may need to make $\mathcal{O}\left(n \cdot m\right)$ calls to this modified version of the FKT algorithm.

**Corollary 2.** *Letting $G$ be a digraph with $n$ vertices and $m$ edges, and letting $\Psi$ be the time complexity of the Jerrum-Sinclair-Vigoda FPRAS [37] for the non-negative matrix permanent, an $\mathcal{O}\left(\Psi \cdot n \cdot m\right)$ time FPAUS exists for KEP-$(\infty, \infty, 0)$ witnesses.*

*Proof.* Observe that the graph $H$ in the Theorem 1 proof argument is guaranteed to be bipartite, and recall that the permanent of a biadjacency matrix for a bipartite graph corresponds to a weighted sum over its perfect matchings. Accordingly, provided that we specify $\Upsilon = 0$, we can again modify the edges of $H$ to have weight 1 and simply replace calls to the FKT algorithm [38–40,58] in the Theorem 1 proof argument with calls to the Jerrum-Sinclair-Vigoda FPRAS [37] for the non-negative matrix permanent. It remains to observe that, as $G$ is not necessarily planar in this context, we need to make at most $\mathcal{O}\left(n \cdot m\right)$ calls to this FPRAS.

**Corollary 3.** *Assuming a IO-split genus $g$ digraph $G$ with $n$ vertices, $m$ edges, and integer edge weights of absolute value $\leq C$, $\forall g \geq 0$ and for $g = 0$ KEP-$(\infty, \infty, \Upsilon)$ can be solved in $\mathcal{O}\left(n^{\frac{1}{2}} \cdot m \cdot \ln\left(n \cdot C\right)\right)$ and $\mathcal{O}\left(n^{\frac{4}{3}} \cdot \ln^2\left(n\right) \cdot \ln\left(n \cdot C\right)\right)$ time, respectively.*

*Proof.* Construct the graph $H$ from $G$ via (steps 1–4) of the Theorem 1 proof argument. Observe that we can utilize the algorithm of Gabow and Tarjan [32], which improves upon the standard Hungarian algorithm [41] modified to use Fibonacci heaps [29], to find a maximum total weight 1-factor of $H$ in time $\mathcal{O}\left(n^{\frac{1}{2}} \cdot m \cdot \ln\left(n \cdot C\right)\right)$. Here, the correspondence between $G$ and $H$ allows us to use this 1-factor to recover a witness for the original instance of KEP-$(\infty, \infty, \Upsilon)$ without an increase in the asymptotic time complexity. Furthermore, as we have that $H$ will be planar if $G$ is bimodal planar, we accordingly have that instances of KEP-$(\infty, \infty, \Upsilon)$ for IO-split genus 0 barter exchange markets can be solved in $\mathcal{O}\left(n^{\frac{4}{3}} \cdot \ln^2(n) \cdot \ln(n \cdot C)\right)$ time via a recent algorithm of Asathulla et al. [8]. It now suffices to observe that the construction of $H$ will take at most $\mathcal{O}(m)$ time, and if $G$ is planar, at most $\mathcal{O}(n)$ time (as a planar graph can have at most $3n - 6$ edges), and that this will not affect the overall time complexities of the aforementioned maximum weight 1-factor algorithms.

## 4 Sampling Algorithms for KEP-$(L_c, L_p, \Upsilon)$ Witnesses Based on Evaluating a $\{-1, 0, 1\}$ Matrix Permanent

For the purposes of this section, we introduce the concept of a *$k$-terminal matchgate* [14, 16, 60–62] which, in this context, we define as follows:

**Definition 3:** *$k$-Terminal matchgate.* Let $\zeta$ be an edge-weighted undirected graph with a specified ordered set of $k$ degree one *terminal vertices* $(\gamma_1, \gamma_2, \ldots, \gamma_k)$. Let $P$ be the set of perfect matchings for $\zeta$. Let $f$ be a function which accepts a binary vector $v \in \{0, 1\}^k$ and returns the subset of perfect matchings $P_v \subseteq P$, where we have that the edge adjacent to the terminal vertex $\gamma_i$ belongs to each perfect matching in the set if and only if the $i$th position in $v$ is 1, and let $h$ be a function which sums over the products of the edge weights in each perfect matching $p_i \in P_v$. Here, letting $\mathcal{S}$ be a set of length $k$ binary vectors, we call $\zeta$ a *$k$-terminal matchgate* with *signature* $\mathcal{S}$ if and only if $\forall v \in \mathcal{S}$ we have that $h(P_v) = 1$ and $\forall v \notin \mathcal{S}$ we have that $h(P_v) = 0$.

We also define the following special type of $k$-terminal matchgate:

**Definition 4:** *$k$-Terminal equality matchgate.* A $k$-terminal matchgate $\zeta$ is a *$k$-terminal equality matchgate* if and only if its signature $\mathcal{S}$ consists of only length $k$ all-0 and all-1 binary vectors.

In Fig. 2(a) we show an instance of a non-bipartite 4-terminal equality matchgate due to Curticapean [16], and in Fig. 2(b) we show our equivalent bipartite realization of the matchgate, where in both cases (thin black), (thick black), and (thick dashed) edges indicate edge weights of 1, $-1$, and $\frac{1}{2}$, respectively. Abstracting the Fig. 2(b) matchgate in the manner shown in Fig. 2(c), we can then make use of the scheme shown in Fig. 2(d) (due to Curticapean [16]) to connect an arbitrary number of copies of this matchgate to generate a bipartite $k$-terminal equality matchgate for any $k \in 2\mathbb{N}_{>0}$ (e.g., see Fig. 2(e)).

We can now proceed to establish the following theorems:

**Theorem 2.** *Assuming access to an idealized coin flipping oracle and letting – (1) $G$ be a digraph with $n$ vertices and $m$ edges; (2) $q_1, q_2, \ldots \in \mathcal{Q}$ be an enumerated set of cycles and paths in $G$ of vertex-wise lengths at most $L_c$ and $L_p$, respectively, where each $q_i \in \mathcal{Q}$ has an associated weight $w_i \in \mathbb{N}_0$; (3) $\mathcal{W}_{\mathcal{Q}}$*

*be the sum of all such weights for elements in $\mathcal{Q}$; (4) $\Lambda$ be the time complexity of an algorithm to calculate the permanent of a $\{-1, 0, 1\}$ square matrix with $\mathcal{O}(n \cdot |\mathcal{Q}| + \mathcal{W}_{\mathcal{Q}})$ columns and rows – an $\mathcal{O}\left(\Lambda \cdot |\mathcal{Q}|^2\right)$ time uniform sampler exists for KEP-$(L_c, L_p, \Upsilon)$ witnesses.*

*Proof.* We will proceed as in the case of the Theorem 1 uniform sampler, though in this context selecting subgraphs in $\mathcal{Q}$ (as opposed to edges in $G$) according to the number of partitions of total weight $\geq \Upsilon$ that they participate in. We will then require the selected subgraph to occur in all partitions (as opposed to simply deleting an edge in the case of the Theorem 1 sampler), and recurse this procedure until the set of fixed subgraphs constitutes a partition of $G$. Accordingly, we require a method of counting the number of partitions of the digraph $G$ into subgraphs from $\mathcal{Q}$, where the sum of the weights of the subgraphs in each partition is $\geq \Upsilon$, and do so by expressing this count as a $\{-1, 0, 1\}$ matrix permanent of a graph $H$.

Here, for every $q_i \in Q$ with $n_i$ vertices, we construct bipartite $(2(n_i + w_i))$-terminal instances of the equality matchgate shown in Fig. 2(d), where the underlying 4-terminal matchgate is shown in Fig. 2(b). Then, for every vertex $v_i$ in $G$, we construct a pair of vertices $\{x_{(i,1)}, x_{(i,2)}\}$, and ensuring that the resulting graph is bipartite, connect these vertices to distinct terminals of the equality matchgates corresponding to the subgraphs $q_i \in \mathcal{Q}$ containing $v_i$. Subsequently, again ensuring that the resulting graph is bipartite, for each $k \in [1, \mathcal{W}_{\mathcal{Q}}]$, we construct a pair of vertices $\{y_{(k,1)}, y_{(k,2)}\}$, and whenever $k > \Upsilon$, add an edge between $y_{(k,1)}$ and $y_{(k,2)}$. Lastly, we connect each $y_{(k,1)}$ and $y_{(k,2)}$ vertex to all equality matchgate terminals in the first and second partite set, respectively, whenever we have that these terminals are not connected to vertices of the form $x_{(i,1)}$ or $x_{(i,2)}$. Observe now that we are simulating the selection of a given subgraph $q_i \in \mathcal{Q}$ by treating the subgraph as a $k$-terminal equality gadget where all terminal vertices are adjacent to an edge in a given perfect matching. This will cover all vertices of the form $x_{(i,\ldots)}$ corresponding to the vertices in the original barter exchange market $G$ covered by $q_i$, ensure that no overlapping subgraphs $q_i, q_j \in \mathcal{Q}$ can be selected simultaneously, and require a selection of a subset of $\mathcal{Q}$ covering all vertices in $G$. Additionally, observe that vertices of the form $y_{(k,\ldots)}$ will ensure that the sum of the weights for this subset of $\mathcal{Q}$ is at least $\Upsilon$. For an explicit example of this construction, we refer the reader to Fig. 3.

Next, we compute an adjacency matrix $\Omega$ for $H$, where we can observe that $\Omega$ will have at most $\mathcal{O}(n \cdot |\mathcal{Q}| + \mathcal{W}_{\mathcal{Q}})$ columns and rows, respectively. Finally, we compute and return the square root of the permanent of $\Omega$, which will correspond to a sum over the number of perfect matchings for $H$. The result will be a count for the number of partitions of the digraph $G$ into subgraphs from $\mathcal{Q}$, where the sum of the weights of the subgraphs in each partition is $\geq \Upsilon$.

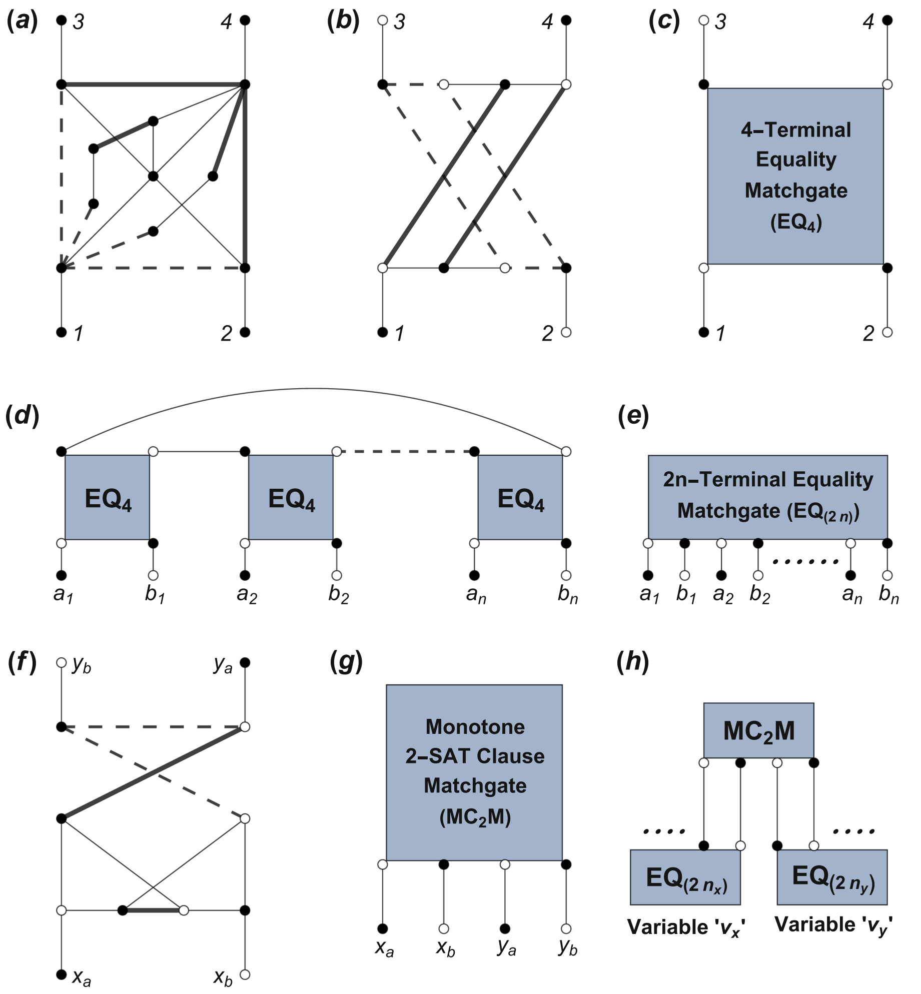


**Fig. 2.** Illustrations of matchgates used in this work and elsewhere: **(a)** non-planar non-bipartite 4-terminal equality ($EQ_4$) matchgate due to Curticapean [16], where (thin black), (thick black), and (thick dashed) edges indicate edge weights of 1, $-1$, and $\frac{1}{2}$, respectively; **(b)** novel bipartite 4-terminal equality ($EQ_4$) matchgate where edge colorations have the same meaning as in (a); **(c)** abstraction of the bipartite $EQ_4$ matchgate from (b); **(d)** scheme to create a $2n$-terminal matchgate $EQ_{2n}$ for any $n \in \mathbb{N}_{>0}$ (identical to one given by Curticapean [16]); **(e)** abstraction of the $EQ_{2n}$ matchgate from (d); **(f)** planar bipartite Monotone 2-SAT Clause Matchgate ($MC_2M$), where edge colorations have the same meaning as in (a); **(g)** abstraction of the $MC_2M$ matchgate from (f); **(h)** partial illustration of the reduction from #Monotone-2-SAT to computing a hollow Hermitian $\{-1, 0, 1\}$ matrix permanent.

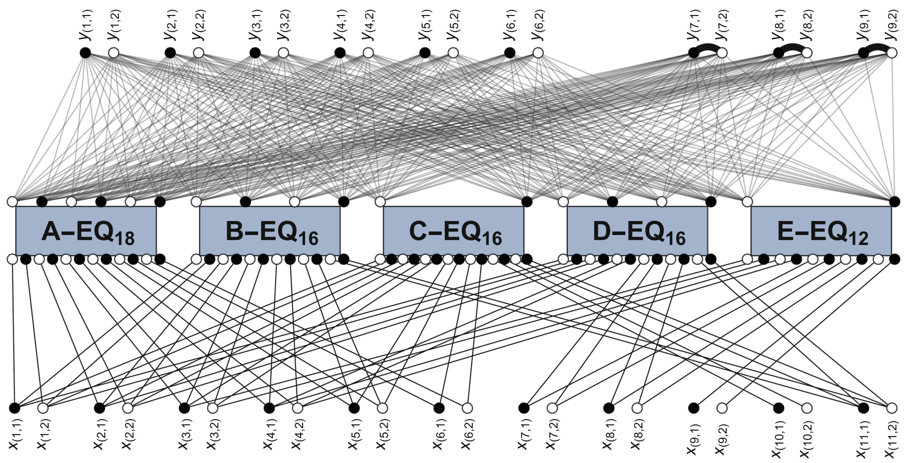


**Fig. 3.** Example of the construction used in Theorem 2 to reduce the problem of counting the number of partitions of a barter exchange market $G$ into weighted subgraphs from a set $\mathcal{Q}$, where a sum over the weights of the subgraphs must be at least equal to $\Upsilon = 6$ (enforced by the top row of vertices); here, $\mathcal{Q}$ corresponds to the subgraphs "A", "B", "C", "D", and "E", which are assigned weights 3, 2, 1, 2, and 1, respectively.

Noting that $\Lambda$ will dominate the time complexity of constructing $H$, and observing that we need to iterate the subgraph selection procedure at most $\mathcal{O}(|\mathcal{Q}|)$ times – each time computing $\mathcal{O}(|\mathcal{Q}|)$ permanents to randomly select a subgraph $q_i \in \mathcal{Q}$, then deleting vertices to force $q_i$ to have only the signature of the all-1 vector – we achieve a uniform random sampler (assuming access to an idealized coin flipping oracle) with the time complexity as stated.

**Theorem 3.** *No FPRAS can exist for computing permanents of hollow Hermitian* $\{-1, 0, 1\}$ *matrices unless* $NP = RP$.

*Proof.* We proceed by giving an AP-reduction from the problem of approximately counting witnesses for instances of Monotone 2-Satisfiability (#Monotone-2-SAT), where #Monotone-2-SAT does not admit an *FPRAS* unless $NP = RP$ [25], to the problem of computing the permanent of a hollow (i.e., all-0 diagonal) Hermitian $\{-1, 0, 1\}$ matrix. Here, we accomplish this by using a bipartite $k$-terminal equality matchgate to encode variables for any instance of #Monotone-2-SAT, and by realizing the bipartite matchgate shown in Fig. 2(f,g), which when connected to the bipartite $k$-input equality gadget in the manner shown in Fig. 2(h), will mimic a 2-SAT clause by requiring at least one of its two pairs of outgoing edges to participate in a perfect matching. We then remove the $\frac{1}{2}$ weights via the scheme detailed in "Remark 1.30" of [16] or "Lemma 7" of [17], which will be of no consequence for our AP-reduction. It remains to observe that the square root of the permanent of the adjacency matrix for any bipartite graph (which will necessarily be a hollow Hermitian matrix) encodes the number of perfect matchings for the graph.

## References

1. Aaronson, S., Arkhipov, A.: The computational complexity of linear optics. In: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC), pp. 333–342. Association for Computing Machinery, New York (2011)
2. Aaronson, S., Arkhipov, A.: The computational complexity of linear optics. Theory Comput. **9**(Article 4), 143–252 (2013)
3. Abraham, D.J., Blum, A., Sandholm, T.: Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In: Proceedings of the 8th ACM conference on Electronic Commerce (EC), pp. 295–304. Association for Computing Machinery, New York (2007)
4. Anderson, R., Ashlagi, I., Gamarnik, D., Roth, A.E.: Finding long chains in kidney exchange using the traveling salesman problem. Proc. Natl. Acad. Sci. U.S.A. **112**(3), 663–668 (2015)
5. AP: Hospitals turn to online matchmakers to swap supplies. Modern Healthcare (2020)
6. Arrazola, J.M., Bromley, T.R.: Using Gaussian boson sampling to find dense subgraphs. Phys. Rev. Lett. **121**, 030503:1–030503:6 (2018)
7. Arrazola, J.M., et al.: Quantum circuits with many photons on a programmable nanophotonic chip. Nature **591**(7848), 54–60 (2021)
8. Asathulla, M.K., Khanna, S., Lahn, N., Raghvendra, S.: A faster algorithm for minimum-cost bipartite perfect matching in planar graphs. ACM Trans. Algorithms **16**(1), 2.1–2.30 (2020)
9. Ashlagi, I., Gamarnik, D., Rees, M.A., Roth, A.E.: The need for (long) chains in kidney exchange. Technical report 18202, National Bureau of Economic Research (NBER), Cambridge, MA (2012)
10. Ashlagi, I., Roth, A.E.: Kidney exchange: an operations perspective. Technical report UTMD-005, University of Tokyo Market Design Center (UTMD), Tokyo, JP (2020)
11. Bondy, J.A., Murty, U.: Graph Theory with Applications, 1st edn. Macmillan Press, New York (1976)
12. BookMooch: (2024). http://bookmooch.com/
13. Brádler, K., Dallaire-Demers, P.L., Rebentrost, P., Su, D., Weedbrook, C.: Gaussian boson sampling for perfect matchings of arbitrary graphs. Phys. Rev. A **98**, 032310:1–032310:15 (2018)
14. Cai, J.Y., Lu, P.: Holographic algorithms: from art to science. J. Comput. Syst. Sci. **77**(1), 41–61 (2011)
15. Coalition, A.: Shoe exchanges (2020). https://www.amputee-coalition.org/resources/shoe-exchanges/
16. Curticapean, R.: The simple, little and slow things count: on parameterized counting complexity. Ph.D. thesis, Saarland University, Saarbrucken, Germany (2015)
17. Curticapean, R.: Parity separation: a scientifically proven method for permanent weight loss. In: Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP), pp. 47:1–47:14 (2016)
18. Curticapean, R., Xia, M.: Parameterizing the permanent: genus, apices, minors, evaluation mod 2k. In: Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 994–1009. IEEE Computer Society, Washington, DC (2015)
19. Deng, Y.H., et al.: Solving graph problems using Gaussian boson sampling. Phys. Rev. Lett. **130**(19), 190601:1–190601:7 (2023)

20. Dickerson, J.P., Procaccia, A.D., Sandholm, T.: Optimizing kidney exchange with transplant chains: theory and reality. In: Proceedings of the 11th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), pp. 711–718 (2012)
21. Dickerson, J.P., Procaccia, A.D., Sandholm, T.: Price of fairness in kidney exchange. In: Proceedings of the 13th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), pp. 1013–1020 (2014)
22. Dickerson, J.P., Sandholm, T.: FutureMatch: combining human value judgments and machine learning to match in dynamic environments. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI), pp. 622–628 (2015)
23. Diestel, R.: Graph Theory, 5th edn. Springer, Heidelberg (2017)
24. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.: Fairness through awareness. In: Proceedings of the 3rd Innovations in Theoretical Computer Science conference (ITCS), pp. 214–226 (2012)
25. Dyer, M., Goldberg, L.A., Greenhill, C., Jerrum, M.: The relative complexity of approximate counting problems. Algorithmica **38**(3), 471–500 (2004)
26. Farnadi, G., Babaki, B., Carvalho, M.: Fairness in kidney exchange programs through optimal solutions enumeration. In: AI for Social Good Workshop, pp. 1–5 (2020)
27. Farnadi, G., St-Arnaud, W., Babaki, B., Carvalho, M.: Individual fairness in kidney exchange programs. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI), pp. 11496–11505 (2021)
28. Fatemi, F.: The regulated market for kidneys in Iran. In: Auctions, Market Mechanisms and their Applications: 2nd international ICST conference (AMMA), pp. 62–75 (2011)
29. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
30. Freedman, R., Borg, J.S., Sinnott-Armstrong, W., Dickerson, J.P., Conitzer, V.: Adapting a kidney exchange algorithm to align with human values. Artif. Intell. **283**(103261), 1–14 (2020)
31. Fresco, A.: Chapter and verse of online book swaps. The Times (2006)
32. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for network problems. SIAM J. Comput. **18**(5), 1013–1036 (1989)
33. Galluccio, A., Loebl, M.: On the theory of Pfaffian orientations. I. Perfect matchings and permanents. Electron. J. Combin. **6**(R6), 1–18 (1999)
34. Hamilton, C.S., Kruse, R., Sansoni, L., Barkhofen, S., Silberhorn, C., Jex, I.: Gaussian boson sampling. Phys. Rev. Lett. **119**, 170501:1–170501:5 (2017)
35. HRSA: Detailed description of data; Figure 1: Patients on the waiting list vs. transplants performed by organ (2020) (2021). https://www.organdonor.gov/learn/organ-donation-statistics/detailed-description
36. Jerrum, M.: Counting, sampling and integrating: algorithms and complexity, chap. Sampling and counting. Lectures in Mathematics, ETH Zuerich, Birkhauser Verlag, Basel, Switzerland (2013)
37. Jerrum, M., Sinclair, A., Vigoda, E.: A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. J. ACM **51**(4), 671–697 (2004)
38. Kasteleyn, P.W.: The statistics of dimers on a lattice: I. The number of dimer arrangements on a quadratic lattice. Physica **27**(12), 1209–1225 (1961)
39. Kasteleyn, P.W.: Dimer statistics and phase transitions. J. Math. Phys. **4**(2), 287–293 (1963)

40. Kasteleyn, P.W.: Graph theory and crystal physics. In: Harary, F. (ed.) Graph Theory and Theoretical Physics, pp. 43–110. Academic Press, London (1967)
41. Kuhn, H.W.: The Hungarian method for the assignment problem. Nav. Res. Logist. Q. **2**(1–2), 83–97 (1955)
42. Kwak, J.Y., Kwon, O.J., Lee, K.S., Kang, C.M., Park, H.Y., Kim, J.H.: Exchange-donor program in renal transplantation: a single-center experience. Transplant. Proc. **31**(1–2), 344–345 (1999)
43. Lund, A.P., Laing, A., Rahimi-Keshari, S., Rudolph, T., O'Brien, J.L., Ralph, T.C.: Boson sampling from a Gaussian state. Phys. Rev. Lett. **113**, 100502:1–100502:5 (2014)
44. Madsen, L.S., et al.: Quantum computational advantage with a programmable photonic processor. Nature **606**(7912), 75–81 (2022)
45. Mattei, N., Saffidine, A., Walsh, T.: Fairness in deceased organ matching. In: Proceedings of the 2018 AAAI/ACM conference on AI, Ethics, and Society (AIES), pp. 236–242 (2018)
46. Newman, J.E., Vardi, M.Y.: FPRAS approximation of the matrix permanent in practice, 38 p. (2020). https://arxiv.org/abs/2012.03367
47. (N.O.S.E.), N.O.S.E.: (2024). http://www.oddshoe.org/
48. Regge, T., Zecchina, R.: Combinatorial and topological approach to the 3D Ising model. J. Phys. A: Math. Gen. **33**(4), 741–761 (2000)
49. Roth, A.E., Sonmez, T., Unver, M.U.: Kidney exchange. Q. J. Econ. **119**(2), 457–488 (2004)
50. Roth, A.E., Sonmez, T., Unver, M.U.: Pairwise kidney exchange. J. Econ. Theory **125**(2), 151–188 (2005)
51. Roth, A.E., Sonmez, T., Unver, M.U.: Efficient kidney exchange: coincidence of wants in markets with compatibility-based preferences. Am. Econ. Rev. **97**(3), 828–851 (2007)
52. Roth, A.E., Sonmez, T., Unver, M.U., Delmonico, F.L., Saidman, S.L.: Utilizing list exchange and nondirected donation through 'chain' paired kidney donations. Am. J. Transplant. **6**(11), 2694–2705 (2006)
53. Saidman, S.L., Roth, A.E., Sonmez, T., Unver, M.U., Delmonico, F.L.: Increasing the opportunity of live kidney donation by matching for two- and three-way exchanges. Transplantation **81**(5), 773–782 (2006)
54. Schnorr, C.P.: Optimal algorithms for self-reducible problems. In: Proceedings of the 3rd International Colloquium on Automata, Languages, and Programming (ICALP), pp. 322–337 (1976)
55. Schnorr, C.P.: On self-transformable combinatorial problems. In: Konig, H., Korte, B., Ritter, K. (eds.) Mathematical Programming at Oberwolfach. Mathematical Programming Studies, vol. 14, pp. 225–243. Springer, Heidelberg (1981)
56. Sempere-Llagostera, S., Patel, R.B., Walmsley, I.A., Kolthammer, W.S.: Experimentally finding dense subgraphs using a time-bin encoded Gaussian boson sampling device. Phys. Rev. X **12**, 031045:1–031045:12 (2022)
57. St-Arnaud, W., Carvalho, M., Farnadi, G.: Adaptation, comparison and practical implementation of fairness schemes in kidney exchange programs (2022). https://arxiv.org/abs/2207.00241
58. Temperley, H., Fisher, M.E.: Dimer problem in statistical mechanics - an exact result. Philos. Mag. **6**(68), 1061–1063 (1961)
59. Tesler, G.: Matchings in graphs on non-orientable surfaces. J. Comb. Theory. Ser. B **78**(2), 198–231 (2000)
60. Valiant, L.G.: Quantum circuits that can be simulated classically in polynomial time. SIAM J. Comput. **31**(4), 1229–1254 (2002)

61. Valiant, L.G.: Holographic algorithms (extended abstract). In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 306–315. IEEE Computer Society, Washington, DC (2004)
62. Valiant, L.G.: Holographic algorithms. SIAM J. Comput. **37**(5), 1565–1594 (2008)
63. Wang, H., et al.: Boson sampling with 20 input photons and a 60-mode interferometer in a $10^{14}$-dimensional Hilbert space. Phys. Rev. Lett. **123**, 250503:1–250503:7 (2019)
64. Wang, X.W., et al.: Experimental boson sampling enabling cryptographic one-way function. Phys. Rev. Lett. **130**, 060802:1–060802:6 (2023)
65. Yoon, D., Lieber, D.: World's first Covid-19 vaccine swap sends Israel's expiring supply to South Korea. Wall Street J. (2021)
66. Zhong, H.S., et al.: Quantum computational advantage using photons. Science **370**(6523), 1460–1463 (2020)

# Author Index

D. Du et al. (Eds.): COCOA 2024, LNCS 15435, pp. 187–189, 2025.
https://doi.org/10.1007/978-981-96-4448-3