Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Automatic Control Laboratory

Master's Thesis

Data-Driven Robust Congestion Pricing

Yize Wang
June 27, 2021

Advisors
Prof. Dr. Dario Paccagnan
Prof. Dr. John Lygeros

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Data-Driven Robust Congestion Pricing

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Wang	Yize

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Harbin, June 2, 2021	Yize Wang

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ETH-Bibliothek
Research Collection

ETH Zürich
Raemistrasse 101
8092 Zurich

Phone +41 44 632 72 22
research-collection@library.ethz.ch
www.library.ethz.ch

Declaration of consent

for publishing or archiving a master thesis, bachelor thesis or student paper on the Research Collection, ETH Zurich's publication platform.

ETH Zurich's publication platform allows users to publish or archive master theses, bachelor theses and student papers written at ETH Zurich. This offers students and graduate students the opportunity to present their work worldwide and/or have it stored permanently in ETH Zurich's IT infrastructure. The published theses fulfil the specified formal quality criteria.

Declaration of the author
I hereby declare that I consent to the ETH Library (please check)

- [x] making my master thesis, bachelor thesis or student paper available to the public via the Research Collection
- [x] archiving my master thesis, bachelor thesis or student paper and publicly displaying its metadata on the Research Collection

The rights of third parties are not infringed by this publication. I consent to any subsequent necessary conversions into other data formats being made.

Author:	Yize Wang
Title of the publication:	Data-Driven Robust Congestion Pricing
Department:	Information Technology and Electrical Engineering
Publication year:	2021
Email/Phone:	yizwang@student.ethz.ch
Private address:	Harbin, China

Zurich, on 02.06.2021 Signature Yize Wang

Recommendation of the responsible ETH professor or research supervisor (please check)

- [x] I have supervised this master thesis, bachelor thesis or student paper at ETH Zurich and recommend its **publication**. In particular, I hereby declare that the publication of this thesis does not infringe the rights of third parties and that any rights to confidentiality are protected.
- [ ] I have supervised this master thesis, bachelor thesis or student paper at ETH Zurich and agree that the ETH Library a**rchives the document and publicly displays its metadata**.

Name Ioannis Lygeros

Zurich, on 06.06.2021 Signature

Declaration of the ETH-Bibliothek
The ETH-Bibliothek guarantees the long-term availability of the thesis as well as the integrity of its content and its authenticity and will make the document accessible via the Internet. The intellectual property rights remain with the author.

Abstract

Self-interested routing is often inefficient in traffic networks. Economists and computer scientists have proven that imposing proper tolls on congested roads, i.e., congestion pricing, can effectively improve network efficiency by reducing the total travel time experienced by all users. However, most researches focus on deterministic demands, while uncertainties are rarely studied. As a result, the tolls designed for the deterministic demands may behave undesirably when the actual demands deviate from the expectation. Existing solutions for uncertain demands mostly require a particular structure of either the network or the uncertainty, which motivates us to investigate a robust congestion pricing scheme for general networks that guarantees the toll performance under unknown uncertainties.

This thesis combines congestion pricing and scenario approach, a general framework for data-driven robust optimization, and formulates the robust toll design task as a bi-level optimization. Taking observed scenarios into account, we are able to design robust tolls that perform well when actual demands respect the same distribution with no knowledge of the distribution required. Besides exact algorithms, we also propose approximate ones to solve the NP-hard bi-level optimization within a reasonable time. We illustrate that the robustness can be quantified, and the toll-setter can compromise between robustness and performance. Finally, we demonstrate with numerical examples that the designed tolls can deal with uncertainties in unseen scenarios.

Keywords Congestion pricing, data-driven decision making, scenario approach, game theory, robust optimization

Acknowledgment

First, I would like to thank Prof. Dr. Dario Paccagnan for supervising me during the unforgettable six-month research. His insightful advice, broad knowledge, patient instruction have always encouraged me to explore more and further. The research topic was a completely new field for me, but with Dario's invaluable help, I managed to learn relevant knowledge quickly and make some contributions to the scientific society. He not only supported me through numerous meetings and discussions but also exerted an imperceptible influence on my personality, with his persistence, humility, optimism and decisiveness. It is my greatest honor to have Dario as my supervisor, and my greatest regret to not be able to meet Dario in person due to the pandemic of COVID-19.

I also would like to thank Prof. Dr. John Lygeros. Among all lectures I had in my life, no one intrigues me more than *Linear System Theory*. John's lecture serves as a perfect bridge between mathematics and engineering, connecting my past knowledge. He also brought me the opportunity to know Dario and to get involved in the enjoyable research.

Furthermore, I would like to thank Dr. Kiril Solovey for providing me with a fast implementation of the traffic-assigning algorithm. The implementation sped up my algorithm more than ten times and made it possible to conduct experiments on huge networks. I would like to thank Prof. Dr. Marco Claudio Campi for proposing the wonderful scenario approach framework. Without his prior work on the scenario approach, my thesis would be impossible. I would also like to thank Yilun Zhang for selflessly sharing his knowledge on optimization theory and numerical solver. Frequent discussions with him greatly encouraged me during the unexpected lockdown.

Moreover, I have my warmest gratitude to my family, Wenying Zhu, Aoming Liu, Chao Jing, Chenyu Zhou, Gaoxiang Shen, Guohui Gao, Hao Yu, Huimin Zhang, Jingyang Shen, Lanyi Yang, Linfeng Xu, Mafu Zhang, Modi Liu, Qi Zeng, Tianwei Lan, Wencan Huang, Xiang Li, Xingyun Zhou, Xinyu Sun, Yan Wu, Yimeng Lu, Yu Dai, Yukai Lin, Yunzhe Pan and Zepeng Shao. Without their company, I cannot overcome all the difficulties and accomplish my thesis.

Finally, I would like to thank Mr. Lixin Tang, ETH Zurich, and China Scholarship Council for financially supporting my study.

No matter what difficulties and challenges we face, as a community with a shared future, humankind should abandon prejudice and unite and move forward in the long course of history.

Abbreviations

OD	Origin-Destination
NP	Nondeterministic Polynomial Time
BPR	U.S. Bureau of Public Roads
OPT	Optimization
SA	Scenario Approach
UE	User Equilibrium
SO	Social Optimum
NLP	Non-Linear Programming
LLP	Lower-Level Programming
BLP	Bi-Level Programming
PoA	Price of Anarchy
MC	Marginal Cost
MCT	Marginal-Cost Tolls
Iter	Iteration
TH	Threshold
VFW	Vanilla Frank-Wolfe Algorithm
CFW	Conjugate Frank-Wolfe Algorithm
BFW	Bi-Conjugate Frank-Wolfe Algorithm
N/A	Not Available
LP	Linear Programming
QP	Quadratic Programming
MILP	Mixed-Integer Linear Programming
MIQP	Mixed-Integer Quadratic Programming
SLSQP	Sequential Least SQuares Programming

Contents

Chapter 1

Introduction

As a result of crowded megacities, the rapid growth of automotive vehicles, and inadequate transportation facilities, traffic congestion has been globally considered as a severe social problem, which causes uncountable waste of time and money [35, 39]. One of the main reasons for such traffic inefficiency is the selfish behavior of traffic participants [16, 15, 38]. That is, the social transportation cost is worse than the optimal one due to participants' tendency to minimize their individual travel costs. The traffic flow resulting from selfishness is commonly modeled as a Wardrop equilibrium and has been extensively studied, for example in [2, 37, 46]. The Wardrop model describes the traffic assignment as a non-atomic non-cooperative game, with each road equipped with a convex latency function. There is a set of origin-destination pairs, called commodities, and each commodity has an amount of traffic flow to deliver, called demand. Given the network structure and commodities, the Wardrop equilibrium can be obtained through convex programming [2].

In order to mitigate the inefficiency caused by selfish behavior, researchers proposed congestion pricing schemes to incentivize rational individuals to behave in a socially-desirable manner. For instance, by imposing tolls on roads, some drivers are discouraged from using the overcrowded links and deviate to previously-unfavored ones, leading to a better social traffic flow [33, 2].

The seminal work [2] proves that charging marginal-cost tolls on all roads will ensure the resulting Wardrop equilibrium coincides with the socially optimal allocation. However, three main issues are prohibiting its application in real-world traffic systems. First, all links are required to be taxable for marginal-cost tolls to take effect. Although highways can be easily taxed, it is currently impossible to impose tolls on every single community street. Second, the value of margin-cost tolls may be arbitrarily large. Third, commodities and their corresponding demands are assumed to be known exactly, while they actually vary constantly and cannot be pre-measured. As a consequence, the flow-independent margin-cost taxation can even exaggerate congestion due to the mismatch between the actual and predicted demands, i.e., not robust to uncertainties [5, 45].

To tackle the first two problems, we can utilize a bi-level optimization, where the upper level optimizes the social cost by choosing bounded tolls, and the lower level describes the Wardrop equilibrium determined by the upper-level decision variables. The bi-level optimization has been proven to be NP-hard and thus is computationally demanding. There are many techniques proposed to solve the bi-level optimization, for example [14, 40, 30, 1]. However, in the worst case, the complexity of mathematical programming grows exponentially with the network scale and can quickly incur an infeasible computational cost.

To handle the uncertainty problem, we resort to the robust optimization framework - scenario approach. The scenario approach is regarded as a powerful methodology for data-driven decision making [7, 8]. The algorithm first collects some recorded scenarios and looks for the solution that optimizes the objective function in the worst case of these scenarios. Interestingly, the scenario theory probabilistically guarantees the solution's performance when the decisions are applied to unseen scenarios. [11] even generalizes the scenario approach theory to consider non-convex optimizations.

In this thesis, we propose several algorithms to compute *probabilistically robust optimal restricted tolls* by combining the bi-level optimization and the scenario approach. We utilize the mathematical programming solver Gurobi to locate the robust globally optimal solution for small networks. For large networks where obtaining the global one is difficult, we present a numerical gradient-descent algorithm to compute suboptimal tolls efficiently. We remark that even the solutions are computed approximately, the scenario approach allows us to probabilistically guarantee the performance under uncertainties.

1.1 Related Work

Researchers have investigated many tolling mechanisms to deal with the inefficiency of the equilibrium in the Wardrop model. [25, 3, 26, 24] compute the optimal tolls for parallel-arc single-commodity networks with affine latency functions where only subnetworks are taxable. [3] shows that toll upper bounds can be respected, and [24] generalizes the type of latency functions. [4] derives the optimal bounded tolls and the best-possible performance guarantee as a function of toll upper bounds. However, [25] shows that if taxable edges form a strict subset of all edges, it would be NP-hard to compute optimal taxes for general networks even with affine latency functions and two commodities.

In the case of general networks and multi-commodities, [24] proposes three heuristic algorithms to arrive at approximately optimal tolls. [43] adopts a numerical method to compute optimal tolls for general subnetworks. With strong assumptions, [12] simplifies the problem into a single-level convex optimization by assuming that the same arcs will be used before and after they are taxed. [18] deals with multi-commodity series-parallel networks numerically via longest-path-first flow decomposition. [29, 30, 49, 50, 47, 48, 17] tackle the tolling problem as a bi-level optimization. However, [27] proves that the simple version of the bi-level problem with linear objective functions and linear constraints is already NP-hard. Even worse, checking whether a solution is locally optimal has also been proved to be NP-hard [44].

When the uncertainty of the demands is taken into account, [13] studies conditions under which the optimal tolls are demand-independent given all links are taxable. [21] proposes a numerical demand-robust tolling mechanism for general networks. [22] presents two approximation methods to compute robustly optimal tolls.

The scenario approach is proposed in [6, 7, 8], which illustrate how the probabilistic robustness can be achieved for convex programs, and [10] steps further to consider relaxing the constraints in order to comprise between the performance and the robustness. The author's later works [11, 9] generalize the theory to non-convex cases and argue that the robustness cannot be evaluated until the non-convex programs are solved.

In this thesis, we consider general traffic networks with multiple commodities. Integrating the scenario approach, we propose both exact and approximate algorithms to compute probabilistically-robust tolls based on seen scenarios. All algorithms can be applied to partially-taxable networks and respect prescribed toll upper bounds. The approximate algorithm can find an approximate local optimum of the NP-hard toll-designing optimization in a reasonable time. Finally, we demonstrate with numerical examples that the designed tolls perform well in unseen scenarios, and the policymaker can compromise between performance and robustness.

Chapter 2

Formulation

2.1 Traffic Assignment Model

2.1.1 Network Model

Consider a directed traffic network $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. $\mathbb{V}$ and $\mathbb{E}$ are the sets of vertices and edges with cardinality of N_V and N_E, respectively. A directed edge is represented as $e = (v_1, v_2)$ starting from vertex v_1 and terminating at vertex v_2. We then associate network $\mathbb{G}$ with a commodity set $\mathbb{C}$, which has N_K origin-destination pairs, $\mathbb{C} = \big((o^1, d^1), \ldots, (o^{N_K}, d^{N_K})\big)$. For i^{th} commodity (o^i, d^i), we have to deliver a certain value of traffic flow, called demand and denoted by d^i. We further represent the traffic flow on edge e caused by i^{th} commodity as f_e^i. Let f_e collect all traffic flows on edge e of all commodities,

$$f_e = \sum_{i=1}^{N_K} f_e^i. \tag{2.1}$$

A flow is *feasible* for the network and the commodity set if for each commodity:

1. the net flow of this commodity out of the origin is equal to the demand;
2. the net flow of this commodity into the destination is equal to the demand;
3. the net flow of this commodity out of other vertices is 0.

These conditions are mathematically formulated as

$$\sum_{e:o=v} f_e^i - \sum_{e:d=v} f_e^i = \begin{cases} d^i & \text{if } v = o^i \\ -d^i & \text{if } v = d^i \\ 0 & \text{otherwise} \end{cases}, \forall i \in \{1, \ldots, N_K\}, \tag{2.2}$$

where o^i and d^i are the origin and destination of the i^{th} commodity (o^i, d^i).

We additionally require flows to be *non-negative* on all directed edges, i.e.,

$$f_e^i \geq 0, \forall e \in \mathbb{E}, i \in \{1, \ldots, N_K\}. \tag{2.3}$$

Next we impose a common assumption before elaborating our traffic model.

Assumption 1 (Existence of Feasible Flow)**.** *There exists at least one feasible flow for network* $\mathbb{G}$ *and commodity set* $\mathbb{C}$*.*

For the ease of notation, we define flow vector $\boldsymbol{f}^k \in \mathbb{R}_+^{N_E}$, whose j^{th} entry describes the traffic flow on the j^{th} edge of the k^{th} commodity. Finally, we declare collective edge flow vector $\boldsymbol{f}_{\mathbb{E}} \in \mathbb{R}_+^{N_E}$ and overall flow vector $\boldsymbol{f} \in \mathbb{R}_+^{N_E+N_E N_K}$

$$\boldsymbol{f}_{\mathbb{E}} = \begin{bmatrix} f_1 \\ \vdots \\ f_{N_E} \end{bmatrix}, \qquad \boldsymbol{f}^i = \begin{bmatrix} f_1^i \\ \vdots \\ f_{N_E}^i \end{bmatrix}, \qquad \boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_{\mathbb{E}} \\ \boldsymbol{f}^1 \\ \vdots \\ \boldsymbol{f}^K \end{bmatrix}, \qquad \boldsymbol{f}_{\mathbb{E}} = \sum_{k=1}^{N_K} \boldsymbol{f}^k. \tag{2.4}$$

Now, by properly defining matrix $\boldsymbol{A} \in \mathbb{Z}^{(N_E+N_K N_V)\times(N_E+N_K N_E)}$ and vector $\boldsymbol{b} \in \mathbb{R}^{N_E+N_K N_V}$, we can compactly express (2.1, 2.2, 2.3) in linear form

$$\begin{aligned} \boldsymbol{A}\boldsymbol{f} &= \boldsymbol{b}, \\ \boldsymbol{f} &\geq \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{A}$ and $\boldsymbol{b}$ are usually sparse.

Note that the network model defined here is called the *link-based formulation* while the other type is the *path-based formulation*. Since in the later one, the number of flow variables grows exponentially with respect to the number of links while in the link-based one linearly, we will adopt the link-based formulation throughout.

2.1.2 Cost Model

We consider edges as congestible resources, and the players have to pay some latency costs when utilizing them. The cost is determined by the load-dependent latency function l_e, a non-negative, differentiable, and non-decreasing convex function. l_e is a function of cumulative flows of all commodities on the edge e, that is, f_e. The social cost incurred on edge e is defined as $f_e l_e(f_e)$, i.e., flow times latency. Summing the costs from all edges yields the total social cost

$$C_{soc} = \sum_e f_e l_e(f_e). \tag{2.5}$$

On the other hand, the individual cost $f_e^i l_e(f_e)$ incurred on edge e depends on the collective flow f_e and the i^{th}-commodity flow f_e^i. The total individual cost for the i^{th} commodity is

$$C_{ind}^i = \sum_e f_e^i l_e(f_e).$$

If drivers are charged traffic tolls for using some links, the individual cost should also include the monetary discouragement and be generalized as

$$\tilde{C}_{ind}^i = \sum_e f_e^i \left(l_e(f_e) + s^i \tau_e \right), \tag{2.6}$$

where τ_e is the toll charged for using link e, and the player sensitivity s^i indicates how much the i^{th} commodity values the cost of one unit of time (latency) compared to one unit of money (toll). For simplicity, we assume players from all commodities share the same sensitivity and drop s^i in (2.6) by including the sensitivity into the toll.

Assumption 2 (Homogeneous Players)**.** *All players from all commodities share the same sensitivity, that is,* $s^1 = \cdots = s^K$.

2.1.3 Latency Model

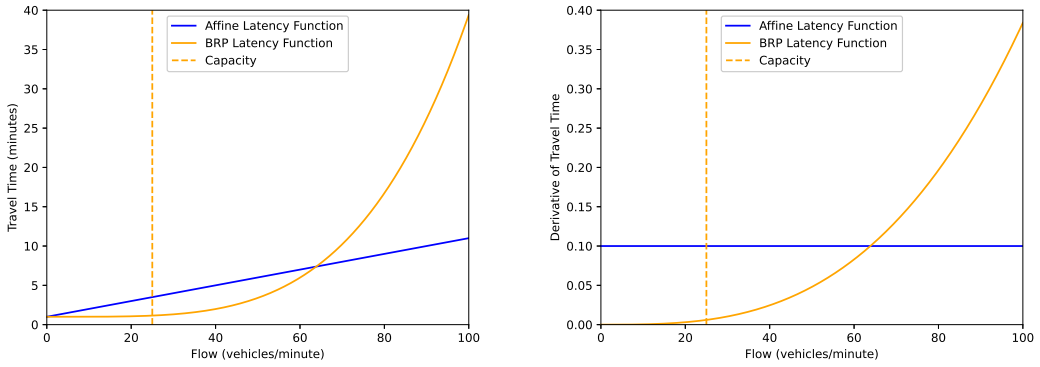
We require latency functions to be non-decreasing, non-negative, differentiable and convex. Among many possible function types, the *affine latency function* is one of the most popular choices in the literature because of its mathematical simplicity, which assumes that the traffic latency grows linearly to the traffic flow on a road.

Definition 3 (Affine Latency Function)**.** *The affine latency function* $l : \mathbb{R}_+ \to \mathbb{R}_+$ *admits the form* $l(f) = af + b$ *with non-negative coefficients* a *and* b*.*

Networks equipped with affine latency functions have been widely investigated in [37, 25, 3, 26, 24]. However, such functions cannot properly reflect how real traffic systems work, because the latency experienced by drivers does not simply grow linearly to the traffic load. Instead, the U.S. Bureau of Public Roads proposed a strongly-convex polynomial function to respect the congestion properties of real roads [31].

Definition 4 (BPR Latency Function)**.** *The Bureau of Public Roads latency function* $l : \mathbb{R}_+ \to \mathbb{R}_+$ *admits the form* $l(f) = t_0 \left(1 + B \left(\frac{f}{C}\right)^P\right)$ *with non-negative coefficients* t_0*,* B*,* C *and* P*.*

In the BPR latency function, t_0 is the free-flow travel time, f is the flow on the link, and C is the capacity, which is the maximum number of vehicles that can pass through a cross section of this road. This function well approximates the real travel time - when the road is at low occupation ($f \ll C$), the users' travel time is close to the free-flow travel time as users do not affect each other much. But if the road is highly congested ($f \approx C$ or $f > C$), the users will suffer from a quick growth of the travel time as the total on-link traffic flow increases, because the polynomial term stands out. Figure 2.1 illustrates the function values and the growth rates of these two latency function types.



(a) Latency Function Values (b) Latency Function Derivatives

Figure 2.1: Latency Function Comparison. Blue curve: affine latency function with $a = 0.1$ and $b = 1.0$; Orange curve: BRP latency function with $t_0 = 1.0, C = 25, P = 4$ and $B = 0.15$.

Due to the strong convexity and the satisfactory approximation of the real traffic behaviors, we will *only* consider the BRP latency function throughout this thesis.

2.2 Equilibrium and Efficiency

2.2.1 Wardrop Equilibrium

As the fundamental concept of the game theory, *Nash equilibrium* characterizes a set of strategies where no single player can reduce his individual cost by unilaterally altering his strategy. However, in real traffic systems, there are always too many players to be efficiently analyzed. Therefore we utilize the non-atomic model where there are infinitely many players for each commodity. Every single player controls only a negligible portion and thus can be considered to have no impact on other players. This motivates the *Wardrop equilibrium* concept and Wardrop gives two main principles in [46], which can be summarized into the following two definitions.

Definition 5 (User Equilibrium)**.** *The feasible traffic flows satisfying the following property are at Wardrop equilibrium: For each commodity, all non-zero-flow paths share the same cost, which is less than those that would be experienced by a player on any unused path.*

Wardrop equilibrium flows are also referred to as user equilibrium flows because no single agent can unilaterally reduce his cost by choosing another path. The user equilibrium is the result of players' selfishness and thus can be often inefficient for the whole traffic system.

Interestingly, Beckman shows in [2] that the user equilibrium coincides with the solution to the following convex optimization:

$$\min_{\boldsymbol{f}} \sum_{e} \int_0^{f_e} l_e(t)\,\mathrm{d}t \tag{OPT-UE}$$

subject to:

$$\boldsymbol{A}\boldsymbol{f} = \boldsymbol{b}$$
$$\boldsymbol{f} \geq \boldsymbol{0}$$

The intuitive explanation is that the sufficient and necessary optimality conditions of (OPT-UE) coincide with Definition 5. The underlying reason is that the non-atomic non-cooperative game above is a potential game, and the objective function in (OPT-UE) is the potential function. When the potential function is minimized, users are at equilibrium.

Finally, we define *socially optimal* traffic flows.

Definition 6 (Socially Optimal Flow)**.** *The feasible traffic flow $\boldsymbol{f}^*$ that minimizes the overall travel costs for players from all commodities are socially optimal, i.e., $\boldsymbol{f}^*$ minimizes (2.5).*

Mathematically, the socially optimal traffic flow solves the following convex optimization:

$$\min_{\boldsymbol{f}} \sum_{e} f_e l_e(f_e) \tag{OPT-SO}$$

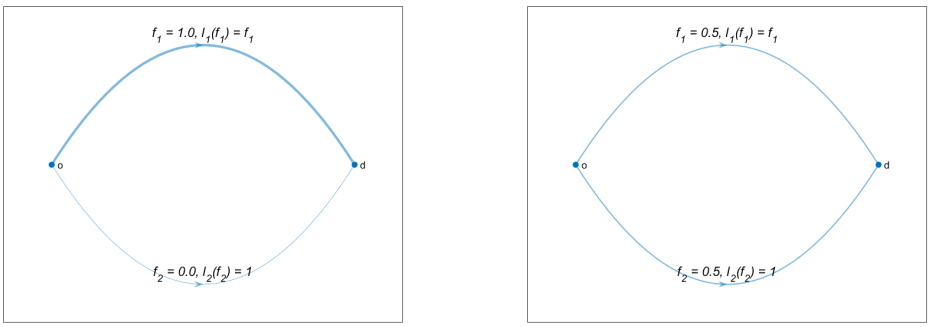
subject to:

$$\boldsymbol{A}\boldsymbol{f} = \boldsymbol{b}$$
$$\boldsymbol{f} \geq \boldsymbol{0}$$

As a result of Assumption 1, the solutions to (OPT-UE) and (OPT-SO) always *exist*. Moreover, because of the strongly-convex objective function and convex constraint set, (OPT-UE) and (OPT-SO) are convex optimizations which admit a *unique* solution.

Theorem 7 (Existence and Uniqueness)**.** *Under Assumption 1, there exists a unique solution to both (OPT-UE) and (OPT-SO).*

To highlight the differences between the UE and SO flows, we introduce the simple Pigou's network. As shown in Figure 2.2, in UE all players choose the upper link because the latency is always less than or equal to 1.0, while in SO half of the players choose the upper one. The social costs of these two cases are 1.0 and 0.75 respectively.



(a) User Equilibrium Flows (b) Socially Optimal Flows

Figure 2.2: User Equilibrium and Socially Optimal Flows in Pigou's Network. There are two nodes in this network: "origin" denoted by o, and "destination" denoted by d. Two edges connect o to d, with latency functions $l_1(f_1) = 1$ and $l_2(f_2) = f_2$. The commodity has a demand of 1.0.

2.2.2 Price of Anarchy

Non-cooperative equilibrium is known to be *inefficient* due to agents' selfish routing tendency, who only want to minimize their individual costs without caring about the social cost [34, 15]. To quantitatively evaluate the inefficiency, [28, 32] introduce the concept of *price of anarchy*.

Definition 8 (Price of Anarchy)**.** *The price of anarchy is the ratio between the worst social cost at a user equilibrium and the optimal social cost, mathematically defined as*

$$PoA = \frac{\max_{\boldsymbol{f} \in \mathbb{F}_{eq}} C_{soc}(\boldsymbol{f})}{\min_{\boldsymbol{f} \in \mathbb{F}} C_{soc}(\boldsymbol{f})},$$

where $\mathbb{F}$ is the set of all feasible flows, and $\mathbb{F}_{eq}$ is the set of all user equilibrium flows.

According to Theorem 7, the price of anarchy can be simplified to the ratio between the social cost of the (unique) user equilibrium flow and the (unique) social optimal flow, that is

$$PoA = \frac{C_{soc}\left(\boldsymbol{f}_{ue}\right)}{C_{soc}\left(\boldsymbol{f}_{so}\right)}.$$

2.3 Toll Design

2.3.1 Deterministic Toll Design by Bi-Level Optimization

The task of designing optimal tolls for a given network and commodities can be intrinsically formulated as a bi-level optimization, where the lower level defines that the flow is at the user equilibrium and the upper level optimizes the social cost over the feasible space of tolls $\mathbb{T}$. We assume that all users value time equally so we can convert monetary cost to time cost. Let $\boldsymbol{\tau}$ denote the time cost converted from the monetary cost. Then, the bi-level optimization reads

$$\min_{\boldsymbol{\tau}\in\mathbb{T}} \sum_e f_e l_e(f_e) \tag{BLP}$$

subject to:

$$\begin{aligned} \boldsymbol{f} = \arg\min_{\boldsymbol{f}} & \sum_e \int_0^{f_e} l_e(t) + \tau_e \, \mathrm{d}t \\ & \boldsymbol{A}\boldsymbol{f} = \boldsymbol{b} \\ & \boldsymbol{f} \geq \boldsymbol{0} \end{aligned} \tag{LLP}$$

For simplicity, we will omit $\mathbb{T}$ in the left part of the thesis. Note that given $\boldsymbol{\tau}$, the lower level is convex in both the objective function and the constraint set, and thus is a convex optimization. To solve (BLP), we substitute (LLP) with KKT conditions,

$$\begin{bmatrix} l_1(f_1) + \tau_1 \\ \vdots \\ l_{N_E}(f_{N_E}) + \tau_{N_E} \\ 0 \\ \vdots \\ 0 \end{bmatrix} - \boldsymbol{\mu} + \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{0} \tag{2.7}$$

$$\boldsymbol{A}\boldsymbol{f} = \boldsymbol{b} \tag{2.8}$$

$$\boldsymbol{f} \geq \boldsymbol{0} \tag{2.9}$$

$$\boldsymbol{\mu} \geq \boldsymbol{0} \tag{2.10}$$

$$\boldsymbol{\mu}^\top \boldsymbol{f} = 0. \tag{2.11}$$

Here, (2.7), (2.8, 2.9), (2.10), (2.11) are called stationarity, primal feasibility, dual feasibility and complementary slackness, respectively. $\boldsymbol{\mu}$ is the dual variable of $\boldsymbol{f}$, and we adopt the same superscripts and subscripts for $\boldsymbol{\mu}$ as $\boldsymbol{f}$. For the ease of notation, we represent the gradient of the objective function in (2.7) as $\nabla C(\boldsymbol{f})$ and compress (2.9), (2.10), (2.11) into $\boldsymbol{0} \leq \boldsymbol{f} \perp \boldsymbol{\mu} \geq \boldsymbol{0}$.

With the transformation above, the bi-level optimization (BLP) becomes a single-level non-linear non-convex optimization in the compact form:

$$\min_{\boldsymbol{\tau},\boldsymbol{f},\boldsymbol{\mu},\boldsymbol{\lambda}} \sum_e f_e l_e(f_e) \tag{NLP}$$

subject to:

$$\begin{aligned} & \nabla C(\boldsymbol{f}) - \boldsymbol{\mu} + \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{0} \\ & \boldsymbol{A}\boldsymbol{f} = \boldsymbol{b} \\ & \boldsymbol{0} \leq \boldsymbol{f} \perp \boldsymbol{\mu} \geq \boldsymbol{0} \end{aligned} \tag{2.12}$$

Note that due to the existence of the bi-linear constraint (2.12), the non-convex problem (NLP) is still NP-hard and thus cannot be solved in reasonable time, making the computational expense formidable for large networks. However, if all links are unlimitedly taxable, marginal-cost tolls are known to be a solution to the above optimization [33, 2].

2.3.2 Marginal-Cost Toll

As defined in [42], the marginal cost is the change in the total cost that arises when the quantity produced changes by one unit. In the context of traffic system, the total cost (social cost incurred by one link) is $f_e l_e(f_e)$ and the marginal cost is defined as the derivative of the social cost with respect to the on-link traffic flow f_e, i.e.,

$$\frac{\mathrm{d}\left(f_e l_e(f_e)\right)}{\mathrm{d}f_e} = l_e(f_e) + f_e l_e^{'}(f_e). \tag{MC}$$

The marginal-cost toll is then defined as $f_e l_e^{'}(f_e)$ - the additional term in (MC) compared to the latency function $l_e(f_e)$.

Definition 9 (Marginal-Cost Toll)**.** *The marginal-cost toll for link e with flow f_e and latency function l_e is $f_e l_e^{'}(f_e)$.*

According to Definition 9, the marginal-cost toll for the affine latency function is

$$f_e l_e^{'}(f_e) = a f_e,$$

and for the BPR latency function is

$$f_e l_e^{'}(f_e) = t_0 B P \left(\frac{f_e}{C}\right)^P,$$

where f_e should be the socially-optimal volume of flow.

Next, we show why marginal-cost tolls incentivize the socially optimal flow. First we consider the user equilibrium subject to marginal-cost tolls. Pluging in the marginal-cost tolls into the objective function of (OPT-UE) gives:

$$\begin{aligned}
\sum_e \int_0^{f_e} \tilde{l}_e(t)dt &= \sum_e \int_0^{f_e} \left(l_e(f_e) + f_e l_e^{'}(f_e)\right) \mathrm{d}t \\
&= \sum_e f_e l_e(f_e)\Big|_0^{f_e} \\
&= \sum_e f_e l_e(f_e).
\end{aligned} \tag{2.13}$$

Since (2.13) coincides with the objective function of (OPT-SO) and the constraint sets are the same, we can conclude that the solutions to (OPT-UE) and (OPT-SO) are identical when marginal-cost tolls are imposed, that is, the marginal-cost tolls force the user equilibrium flows to be the socially optimal flows.

Theorem 10. *Marginal-cost tolls incentivize the socially optimal flow when all users value time equally.*

To compute the flow-independent marginal-cost tolls, we can first calculate the socially optimal flow $\boldsymbol{f}^*$, and multiply the flow by the derivative of the latency function, i.e., $f_e^* l_e^{'}(f_e^*)$.

A natural question is "How well marginal-cost tolls can do in the case of uncertain demands" because in reality, we can only estimate the demands instead of knowing them exactly. Unfortunately, the performance of such tolls is not guaranteed under uncertainties [5, 45]. We take an example of the famous Braess's network in Figure 2.3.

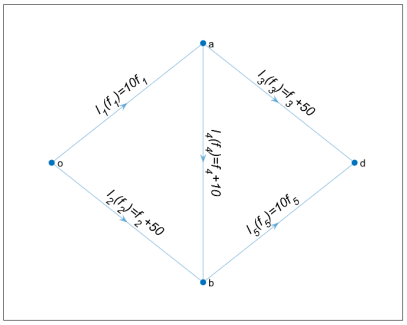
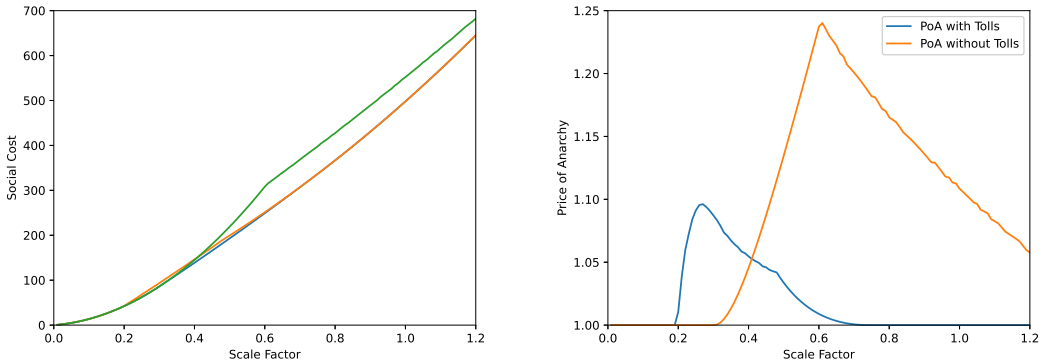


Figure 2.3: Braess's Network. The latency functions are $l_1(f_1) = 10f_1$, $l_2(f_2) = f_2 + 50$, $l_3(f_3) = f_3 + 50$, $l_4(f_4) = f_4 + 10$, $l_5(f_5) = 10f_5$. The demand to be delivered from o to d is 6.0.

Next, we scale the demand and see how social cost curves and price of anarchy curves vary with respect to the scale factor. From Figure 2.4, we observe that the marginal-cost tolls cannot guarantee the improvement of PoA, and can even make the system less efficient under some conditions. This motivates us to design tolls that are *robust* in the face of uncertain demands.



(a) Performance on Social Cost

(b) Performance on Price of Anarchy

Figure 2.4: Performance of Marginal-Cost Tolls. $\tau_1 = 30$, $\tau_2 = 3$, $\tau_3 = 3$, $\tau_4 = 0$, $\tau_5 = 30$.

2.3.3 Scenario Approach

In order to be robust to uncertain demands, we can collect demand data from multiple days and periods, for example, all afternoons in a whole year. The demands may be regarded as random variables which belong to an unknown distribution. The toll design task can now be interpreted as a *data-driven decision making* task.

The scenario approach as a general methodology for the data-driven optimization is well-suited for the task. In this section, we will summarize key assumptions and conclusions in [11]. For detailed derivations, please refer to Campi's studies on the scenario approach [8, 7, 9, 20, 10, 11].

Scenario Approach Setup: Let Δ be a probability space with a probability measure $\mathbb{P}$. An element $\delta \in \Delta$ is called a *scenario*. We call $\left(\delta^{(1)}, \ldots, \delta^{(m)}\right)$ a sample which consists of m independently-drawn scenarios from Δ. Each $\delta^{(i)}$ is regarded as an observation (scenario). Θ is a decision space without any pre-defined properties. The decision maker is given the sample $\left(\delta^{(1)}, \ldots, \delta^{(m)}\right)$ and makes a decision based on a function $\mathcal{A}_m : \Delta^m \to \Theta$. We call the decision $\theta_m^* = \mathcal{A}_m\left(\delta^{(1)}, \ldots, \delta^{(m)}\right)$ the *scenario decision*. A natural feasibility assumption is forced for the scenario approach:

Assumption 11 (Feasibility Assumption)**.** *To every scenario $\delta \in \Delta$, there is an associated constraint set $\Theta_\delta \in \Theta$, which identifies the decisions that are admissible for the situation represented by δ. For all $m = 1, 2, \ldots$ and for any sample $\left(\delta^{(1)}, \ldots, \delta^{(m)}\right)$, it holds that $\mathcal{A}_m\left(\delta^{(1)}, \ldots, \delta^{(m)}\right) \in \Delta_{\delta^{(i)}}$ for all $i = 1, \ldots, m$.*

The robustness of the scenario decision θ_m^* means how well it generalizes to unseen situations $\delta \in \Delta$. We say that θ_m^* *generalizes* to δ if $\theta_m^* \in \Theta_\delta$, otherwise, *violates* δ. Formally, we give the definition of the probabilistic robustness.

Definition 12 (Violation Probability)**.** *The violation probability of a given decision $\theta \in \Theta$ is defined as*

$$\mathbb{V}(\theta) \coloneqq \mathbb{P}\{\delta \in \Delta : \theta \notin \Theta_\delta\}.$$

For a given reliability parameter $\epsilon \in (0, 1)$, we say that $\theta \in \Theta$ is ϵ-feasible if $\mathbb{V}(\theta) < \epsilon$.

A prominent feature of the scenario approach in the non-convex setting is that we cannot assert the robustness until we solve θ_m^*. In other words, the robustness can only be *a posterior* computed, which is also called the *walk-and-judge* scheme. The robustness depends on the cardinality of the support subsample, defined as follows.

Definition 13 (Support Subsample)**.** *Given a sample $\left(\delta^{(1)}, \ldots, \delta^{(N)}\right) \in \Delta^N$, a support subsample $\mathcal{S}$ for $\left(\delta^{(1)}, \ldots, \delta^{(N)}\right)$ is a k-tuple of elements extracted from $\left(\delta^{(1)}, \ldots, \delta^{(N)}\right)$, i.e., $\mathcal{S} = \left(\delta^{(i_1)}, \ldots, \delta^{(i_k)}\right)$ with $i_1 < i_2 < \cdots < i_k$, which gives the same solution as the original sample, that is,*

$$\mathcal{A}_k\left(\delta^{(i_1)}, \ldots, \delta^{(i_k)}\right) = \mathcal{A}_N\left(\delta^{(1)}, \ldots, \delta^{(N)}\right).$$

A support subsample is called *irreducible* if no element can be further removed from $\mathcal{S}$ leaving the solution unchanged.

We define a function $\mathcal{B}_N$ to determine the support subsample. Let $\mathcal{B}_N$ map a sample to the set of support subsample indices, i.e., $\mathcal{B}_N : \left(\delta^{(1)}, \ldots, \delta^{(N)}\right) \mapsto \{i_1, \ldots, i_k\}, i_1 < \cdots < i_k$ and $\left(\delta^{(i_1)}, \ldots, \delta^{(i_k)}\right)$ is a support subsample. Note that neither $\mathcal{B}_N$ nor the support subsample is unique. Finally, we cite the key theorem of the scenario approach, proven in Section III of [11].

Theorem 14. *Suppose the Assumption 11 holds true, and set a value $\beta \in (0,1)$ (confidence parameter). Let $\epsilon : \{0,\dots,N\} \to [0,1]$ be a function such that*

$$\epsilon(N) = 1 \tag{2.14}$$

$$\sum_{k=0}^{N-1} \binom{N}{k} (1-\epsilon(k))^{N-k} = \beta. \tag{2.15}$$

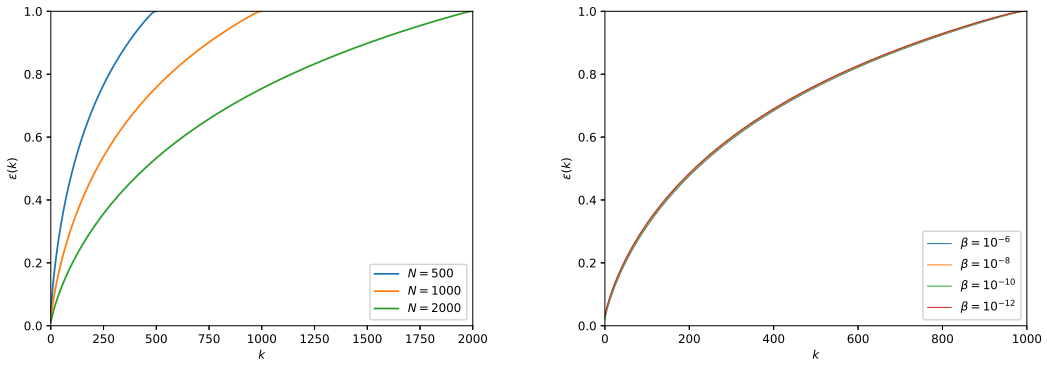
Then, for any $\mathcal{A}_N$, $\mathcal{B}_N$, and probability $\mathbb{P}$, it holds that

$$\mathbb{P}^N\{\mathbb{V}(\theta_N^*) > \epsilon(s_N^*)\} \leq \beta. \tag{2.16}$$

s_N^* in (2.16) means the cardinality of a support subsample. The conclusion (2.16) should be interpreted as "the violation probability of the scenario decision θ_N^* is no larger than $\epsilon(s_N^*)$ with the confidence of $1-\beta$". The confidence parameter β is given by the decision maker. $\epsilon(k)$ defined by (2.14) and (2.15) prescribes a family of functions, and is thus *not unique*. Here, we offer a simple choice of $\epsilon(k)$

$$\epsilon(k) := \begin{cases} 1 & \text{if } k = N, \\ 1 - \sqrt[N-k]{\dfrac{\beta}{N\binom{N}{k}}} & \text{otherwise.} \end{cases} \tag{2.17}$$

In Figure 2.5, we illustrate the impact of the cardinality of the sample N and the confidence parameter β on the function $\epsilon(k)$.



(a) Sample Cardinality's Impact (b) Confidence Parameter's Impact

Figure 2.5: Reliability Parameter

One can easily observe that all four curves in Figure 2.5b are close to each other. This is not a coincidence and is the result of (2.17).

$$\begin{aligned} \epsilon(k) &= 1 - \exp\left(\log\left(\sqrt[N-k]{\frac{\beta}{N\binom{N}{k}}}\right)\right) \\ &= 1 - \exp\left(-\frac{1}{N-k}\log\frac{1}{\beta} - \frac{1}{N-k}\log N\binom{N}{k}\right) \\ &\leq \frac{1}{N-k}\log\frac{1}{\beta} + \frac{1}{N-k}\log N\binom{N}{k} \end{aligned} \tag{2.18}$$

(2.18) shows that $\epsilon(k)$ is logarithmically dependent on β, which indicates that we can ask for a very small β without significantly influencing $\epsilon(k)$.

2.3.4 Robust Toll Design by Scenario Optimization

To design robust tolls based on previous observations, we now resort to the scenario approach. To utilize the scenario approach, we solve the tolls that optimize the *empirical* worst-case social cost over all seen scenarios.

Recall the non-linear formulation (NLP), the demands only affect the vector term $\boldsymbol{b}$. For a given demand vector of scenario i, we highlight the dependence on the scenario by substituting $\boldsymbol{b}$ with $\boldsymbol{b}^{(i)}, i \in \{1, \dots, N_S\}$, where N_S is the number of scenarios. For scenario i, the optimization to minimize the social cost is as follows.

$$\begin{aligned}
&\min_{\boldsymbol{\tau}, \boldsymbol{f}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\lambda}^{(i)}} \sum_e f_e^{(i)} l_e(f_e^{(i)}) \\
&\text{subject to:} \\
&\qquad \nabla C(\boldsymbol{f}^{(i)}, \boldsymbol{\tau}) - \boldsymbol{\mu}^{(i)} + \boldsymbol{A}^\top \boldsymbol{\lambda}^{(i)} = \boldsymbol{0} \\
&\qquad \boldsymbol{A}\boldsymbol{f}^{(i)} = \boldsymbol{b}^{(i)} \\
&\qquad \boldsymbol{0} \le \boldsymbol{f}^{(i)} \perp \boldsymbol{\mu}^{(i)} \ge \boldsymbol{0}
\end{aligned}$$

To minimize the worst-case social cost, we adopt the epigraphical formulation. Let H be the worst-case social cost. We then look for the set of tolls that optimize H.

$$\begin{aligned}
&\min_{H, \boldsymbol{\tau}, \boldsymbol{f}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\lambda}^{(i)}} H \qquad \text{(SA-H)} \\
&\text{subject to:} \\
&\qquad H \ge \sum_e f_e^{(i)} l_e(f_e^{(i)}), \forall i \in \{1, \dots, N_S\} \\
&\qquad \nabla C(\boldsymbol{f}^{(i)}, \boldsymbol{\tau}) - \boldsymbol{\mu}^{(i)} + \boldsymbol{A}^\top \boldsymbol{\lambda}^{(i)} = \boldsymbol{0} \\
&\qquad \boldsymbol{A}\boldsymbol{f}^{(i)} = \boldsymbol{b}^{(i)} \\
&\qquad \boldsymbol{0} \le \boldsymbol{f}^{(i)} \perp \boldsymbol{\mu}^{(i)} \ge \boldsymbol{0}.
\end{aligned}$$

We are interested in solving (SA-H) because the scenario approach provides a quantification of robustness. Since the constraints on $\boldsymbol{\tau}$ are the same for both seen and unseen scenarios, the optimal solution $\boldsymbol{\tau}^*$ to (SA-H) is always admissible for unseen scenarios, which makes Assumption 11 true. Then, the conclusion drawn from (2.16) reads "the probability that the social cost of an unseen scenario is higher than H^* is no bigger than $\epsilon(s_N^*)$ with the confidence of $1 - \beta$, where H^* solves (SA-H)."

When the decision maker is interested in the efficiency instead of the social cost, (SA-H) may be adapted to consider PoA.

$$\min P \tag{SA-P}$$

subject to:

$$P \geq \frac{C_{soc}(\boldsymbol{f}_{ue}^{(i)})}{C_{soc}(\boldsymbol{f}_{so}^{(i)})}$$

$$\boldsymbol{f}_{ue}^{(i)} = \arg\min \sum_e \int_0^{f_{ue,e}^{(i)}} l_e(t) + \tau_e \, \mathrm{d}t, \, \forall i$$

$$\boldsymbol{A}\boldsymbol{f}_{ue}^{(i)} = \boldsymbol{b}^{(i)}$$

$$\boldsymbol{f}_{ue}^{(i)} \geq \boldsymbol{0}$$

$$\boldsymbol{f}_{so}^{(i)} = \arg\min \sum_e f_{so,e}^{(i)} l_e(f_{so,e}^{(i)}), \, \forall i$$

$$\boldsymbol{A}\boldsymbol{f}_{so}^{(i)} = \boldsymbol{b}^{(i)}$$

$$\boldsymbol{f}_{so}^{(i)} \geq \boldsymbol{0}$$

The decision variables in this bi-level optimization are P, $\boldsymbol{\tau}$, $\boldsymbol{f}_{ue}^{(i)}$, and $\boldsymbol{f}_{so}^{(i)}$.

Chapter 3

Algorithms for Globally Optimal Solution

The Gurobi Optimizer is a commercial optimization solver for mathematical programming, which can guarantee the global optimality of the solution for common programming types, for example, mixed-integer linear programming and mixed-integer quadratic programming [23]. Therefore, we will utilize the Gurobi Optimizer to locate the globally optimal solutions for the previously-mentioned optimizations.

3.1 Model Reformulations

We consider the worst-case social-cost optimization (SA-H) and demonstrate the possible reformulations. From (2.4), one can notice that with the non-negativity of the flow vector $\boldsymbol{f}^k$, the collective flow vector $\boldsymbol{f}_{\mathbb{E}}$ is assured to be non-negative. The fact motivates us to remove the bound constraints on $\boldsymbol{f}_{\mathbb{E}}$ and the corresponding complementary slackness constraints.

$$\min_{H,\boldsymbol{\tau},\boldsymbol{f}^{(i)},\boldsymbol{\mu}^{(i)},\boldsymbol{\lambda}^{(i)}} H \tag{SA-H-R1}$$

subject to:

$$\begin{aligned} H &\geq \sum_e f_e^{(i)} l_e(f_e^{(i)}), \forall i \in \{1,\dots,N_S\} \\ &\nabla C(\boldsymbol{f}^{(i)},\boldsymbol{\tau}) - \boldsymbol{\mu}^{(i)} + \boldsymbol{A}^\top \boldsymbol{\lambda}^{(i)} = \boldsymbol{0} \\ &\boldsymbol{A}\boldsymbol{f}^{(i)} = \boldsymbol{b}^{(i)} \\ &\boldsymbol{0} \leq \boldsymbol{f}^{k(i)} \perp \boldsymbol{\mu}^{k(i)} \geq \boldsymbol{0}, \forall k \in \{1,\dots,N_K\}. \end{aligned}$$

By reformulation (SA-H-R1), we remove $3N_E$ constraints in total - N_E constraints for each of the primal feasibility, dual feasibility and complementary slackness. The number of decision variables is $1 + N_E + 2N_E N_S + 2N_K N_E N_S + N_K N_V N_S$. More specifically, H introduces 1 decision variable, $\boldsymbol{\tau}$ introduces N_E, $\boldsymbol{f}^{(i)}$ introduces $(N_E + N_E N_K)N_S$, $\boldsymbol{\mu}^{(i)}$ introduces $N_E N_K N_S$, and $\boldsymbol{\lambda}^{(i)}$ introduces $(N_E + N_K N_V)N_S$.

Although the Gurobi Optimizer is able to deal with the bi-linear constraint (2.12), it is still wise to explicitly linearize this constraint by exploiting the zero-product structure. We achieve the linearization with the big-M notation.

First, we demonstrate with a minimal example $ab = 0, a \geq 0, b \geq 0$, which means either non-negative a or non-negative b is 0 (inclusive or). We introduce an auxiliary *binary* variable $\delta \in \{0, 1\}$ and a large positive number M. Then, consider the following conditions.

$$\begin{aligned} &0 \leq a \leq M\delta \\ &0 \leq b \leq M(1-\delta) \\ &\delta \in \{0, 1\} \end{aligned} \tag{Big-M}$$

(Big-M) reads $a = 0, 0 \leq b \leq M$ if $\delta = 0$, otherwise $0 \leq a \leq M, b = 0$. Since M is a large number, we can relax the upper bound without improperly reducing the search space, that is, $a = 0, b \geq 0$ if $\delta = 0$, otherwise $a \geq 0, b = 0$. Theoretically, we would like M to be $+\infty$ because the relaxation will be strictly equivalent to the original constraints. However, in practice, too large M will cause fatal numerical errors and make the model unstable. Moreover, the default precision of the Gurobi Optimizer is 10^{-6} and with too large M, the solver may terminate undesirably. An empirical choice of M is one or two orders larger than a and b. We now employ the big-M notation to linearize the previous formulation (SA-H).

$$\min_{H, \boldsymbol{\tau}, \boldsymbol{f}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\lambda}^{(i)}, \boldsymbol{\delta}^{(i)}} H \tag{SA-H-R2}$$

subject to:

$$\begin{aligned} H &\geq \sum_e f_e^{(i)} l_e(f_e^{(i)}), \forall i \in \{1, \ldots, N_S\} \\ &\nabla C(\boldsymbol{f}^{(i)}, \boldsymbol{\tau}) - \boldsymbol{\mu}^{(i)} + \boldsymbol{A}^\top \boldsymbol{\lambda}^{(i)} = \mathbf{0} \\ &\boldsymbol{A}\boldsymbol{f}^{(i)} = \boldsymbol{b}^{(i)} \\ &\mathbf{0} \leq \boldsymbol{f}^{(i)} \leq M\boldsymbol{\delta}^{(i)} \\ &\mathbf{0} \leq \boldsymbol{\mu}^{(i)} \leq M\left(\mathbf{1} - \boldsymbol{\delta}^{(i)}\right) \\ &\boldsymbol{\delta}^{(i)} \in \{0, 1\}^{N_E + N_E N_K}, \end{aligned}$$

where $\boldsymbol{\delta}^{(i)}$ is one-to-one related to $\boldsymbol{f}^{(i)}$ and $\boldsymbol{\mu}^{(i)}$, and adopts the same superscripts and subscripts. The number of decision variables is $1 + N_E + 4N_E N_S + 3N_K N_E N_S + N_K N_V N_S$, with 1 for H, N_E for $\boldsymbol{\tau}$, $(N_E + N_E N_K)N_S$ for $\boldsymbol{f}^{(i)}$, $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\delta}^{(i)}$, and $(N_E + N_K N_V)N_S$ for $\boldsymbol{\lambda}^{(i)}$.

Although (SA-H-R2) significantly increases the number of decision variables and constraints, it simplifies the constraint structure by replacing the bi-linear constraints with the linear ones. If affine latency functions are used, (SA-H-R2) turns out to be a mixed-integer quadratically constrained programming, in which the Gurobi Optimizer is specialized.

Finally, we combine the ideas of (SA-H-R1) and (SA-H-R2) to propose a third reformulation.

$$\min_{H,\boldsymbol{\tau},\boldsymbol{f}^{(i)},\boldsymbol{\mu}^{(i)},\boldsymbol{\lambda}^{(i)},\boldsymbol{\delta}^{(i)}} H \tag{SA-H-R3}$$

subject to:

$$\begin{aligned}
&H \geq \sum_e f_e^{(i)} l_e(f_e^{(i)}), \forall i \in \{1,\dots,N_S\} \\
&\nabla C(\boldsymbol{f}^{(i)},\boldsymbol{\tau}) - \boldsymbol{\mu}^{(i)} + \boldsymbol{A}^\top \boldsymbol{\lambda}^{(i)} = \mathbf{0} \\
&\boldsymbol{A}\boldsymbol{f}^{(i)} = \boldsymbol{b}^{(i)} \\
&\mathbf{0} \leq \boldsymbol{f}^{k(i)} \leq M\boldsymbol{\delta}^{k(i)}, \forall k \in \{1,\dots,N_K\} \\
&\mathbf{0} \leq \boldsymbol{\mu}^{k(i)} \leq M\left(\mathbf{1} - \boldsymbol{\delta}^{k(i)}\right), \forall k \in \{1,\dots,N_K\} \\
&\boldsymbol{\delta}^{k(i)} \in \{0,1\}^{N_E+N_E N_K}, \forall k \in \{1,\dots,N_K\}
\end{aligned}$$

The number of decision variables is $1 + N_E + 2N_E N_S + 3N_K N_E N_S + N_K N_V N_S$, with 1 for H, N_E for $\boldsymbol{\tau}$, $(N_E + N_E N_K)N_S$ for $\boldsymbol{f}^{(i)}$, $N_E N_K N_S$ for $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\delta}^{(i)}$, and $(N_E + N_K N_V)N_S$ for $\boldsymbol{\lambda}^{(i)}$. Compared with (SA-H-R1), (SA-H-R3) introduces $N_K N_E N_S$ binary decision variables.

Next, we take the famous Sioux Falls network for example to compare the computational efficiency of these formulations. As shown in Figure 3.1, the network structure and demand data are provided by Ben Stabler [41]. Briefly speaking, there are 24 nodes, 76 edges, and 528 commodities in the Sioux Falls network, and each link is equipped with a BPR latency function.

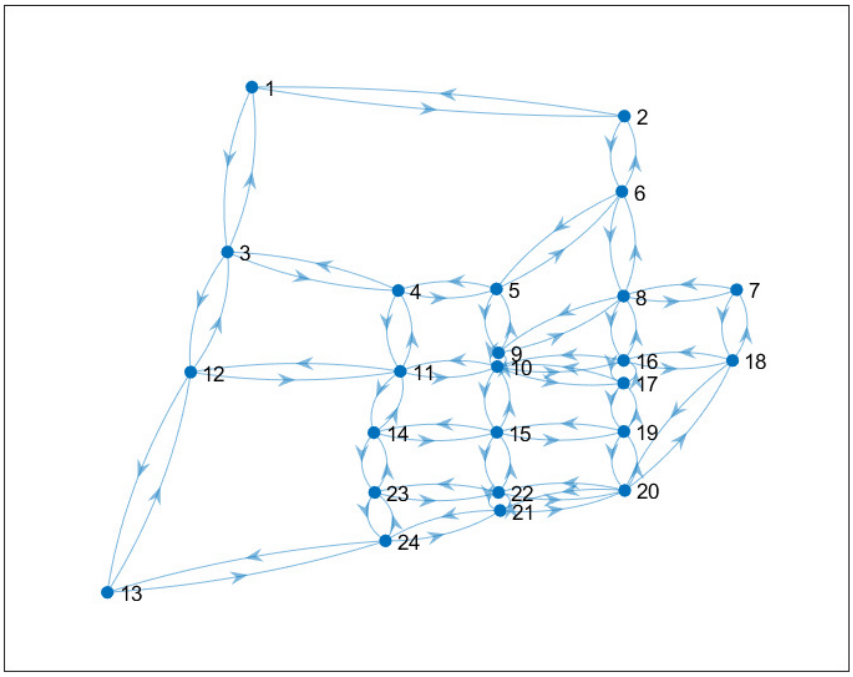


Figure 3.1: Sioux Falls Network

The computational times for only one sample are listed in Table 3.1 and the information of the test environment in Table 3.2. The number of origin-destination pairs (commodities) means how many ODs we take into account. Take **30 ODs** for example, we considered only the first 30 OD pairs. With **528 ODs**, all commodities are considered.

Table 3.1: Computational Time vs Formulations and OD Pair Numbers (Unit: Second)

Formulation	**30 ODs**	**40 ODs**	**50 ODs**	**100 ODs**	**528 ODs**
SA-H	11.12	48.11	37.58	220.61	1765.52
SA-H-R1	16.64	33.70	36.95	191.80	1743.88
SA-H-R2	2.97	9.35	11.50	35.82	918.08
SA-H-R3	1.40	4.04	6.92	14.41	423.85

Table 3.2: Test Environment Information

Item	**Parameter**
Model	Dell XPS 15 9570
Processor	Intel® Core™ i5-8300H CPU @ 2.30GHz
RAM	16.00 GB
System	64-bit Windows 10 Pro
Solver	Gurobi 9.1.2

Table 3.1 shows that all three reformulations can improve the computational efficiency and the reformulation SA-H-R3 performs best.

3.2 Magnitude Scaling

In this section, we address a numerical issue that will appear when the BRP latency functions are adopted. Recall the function form $l(f) = t_0 + t_0 B \left(\frac{f}{C}\right)^P$. The capacity coefficient C presenting in the denominator is usually a huge number. With the common setting $B = 0.15, P = 4$, the coefficient $t_0 B \left(\frac{1}{C}\right)^P$ can be super tiny and even less than the machine epsilon. We have to scale the flow f to overcome the numerical problem. Let C_0 be the scaling factor, the BRP latency function is turned into

$$\begin{aligned} l(f) &= t_0 + t_0 B \left(\frac{f}{C}\right)^P \\ &= t_0 + t_0 B \left(\frac{f C_0}{C C_0}\right)^P \\ &= t_0 + t_0 B \left(\frac{C_0}{C}\right)^P \left(\frac{f}{C_0}\right)^P \\ l(\bar{f}) &\coloneqq t_0 + t_0 B \left(\frac{C_0}{C}\right)^P \bar{f}^P. \end{aligned}$$

where $\bar{f}$ is the scaled flow. Note that C_0 is a constant applied on the BRP latency functions on *all* edges, i.e., C_0 does not change with edges. When C_0 is properly chosen, the numerical issue will be greatly mitigated. For example, in the Sioux Falls network, the coefficient range of the optimization (OPT-UE) is $[2 \times 10^{-9}, 1 \times 10^{1}]$, while $[5 \times 10^{-1}, 5 \times 10^{4}]$ after a proper scaling. Therefore, we replace the flows in the original formulation with the scaled ones to avoid the numerical problem. After the solutions are obtained, we then retrieve the actual flows by multiplying C_0 back, $f = C_0 \bar{f}$.

We offer four intuitive choices of C_0, the minimum, average, median, and maximum of all capacities. Nash flows of the Sioux Falls network are computed with these scaling factors and reported in Table 3.3. We highlight that in spite of the uniqueness of the solution to (OPT-UE), the Gurobi Optimizer returns different results due to the numerical precision.

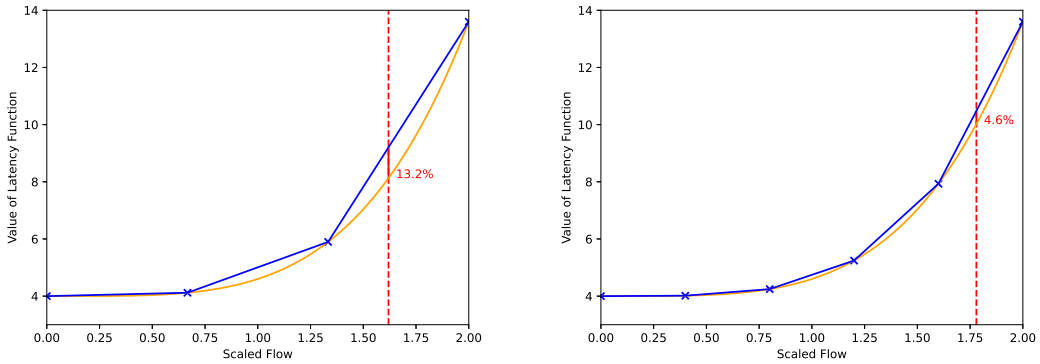
Table 3.3: Scaling Factor Comparison

Variable	$C_0 = \min C$	$C_0 = \text{avg } C$	$C_0 = \text{mid}$	$C_0 = \max C$	GT [1]
f_1	4532.91	4550.17	4529.57	4851.10	4494.66
f_2	8132.91	8150.17	8134.92	8451.10	8119.08
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
f_{75}	10261.12	10231.78	10262.40	10186.65	10259.52
f_{76}	7847.43	7875.60	7843.25	8013.35	7861.83
Optimal Objective	4.23137e+6	4.23141e+6	4.23137e+6	4.23743e+6	4.23134e+6
Relative Error	0.007‰	0.017‰	0.007‰	1.439‰	0.000‰

[1] Edge flows from [41] are taken as the ground truth.

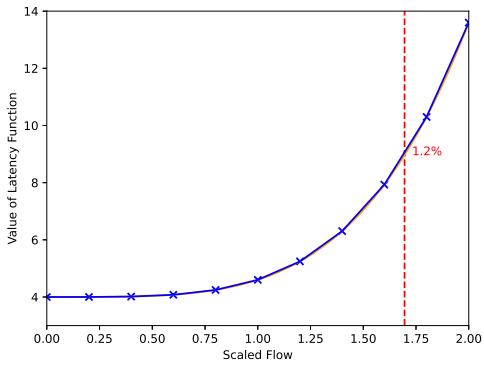
3.3 Approximation for Polynomial Latency Function

The Gurobi Optimizer is specialized in dealing with the linear and quadratic function in the objective function and constraints. Other types of functions, such as polynomial and exponential functions, are approximated by piece-wise linear functions. Take the BRP latency function with the scaled flow for example, Figure 3.2 shows the BRP function and its linear approximation, with the maximal relative difference highlighted. Note that these pieces are linearly-spaced, that is, they share the same length of the projection on the x axis. More sophisticated approximation is possible, for example, the Ramer–Douglas–Peucker algorithm - add pieces where the relative error is greater than a threshold.

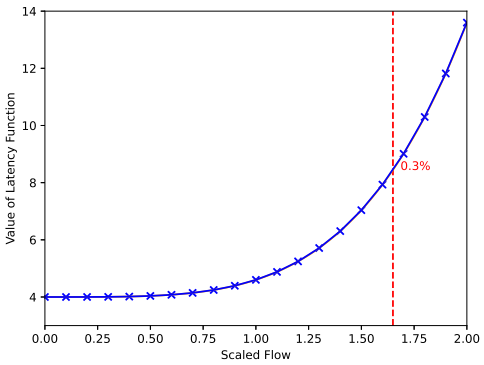


(a) 3 Pieces

(b) 5 Pieces



(c) 10 Pieces



(d) 20 Pieces

Figure 3.2: Piece-wise Linear Approximation with Different Pieces

Figure 3.3 shows that the maximal relative error of the approximation drops quickly as the number of pieces increases, at the cost of the computational efficiency.

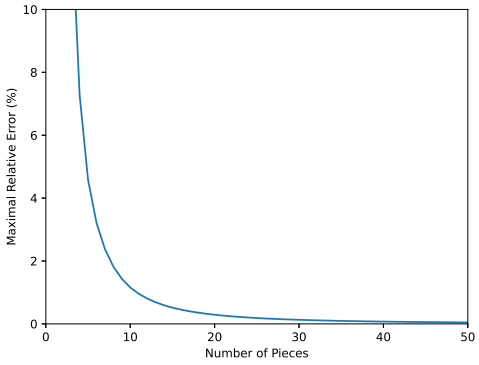


Figure 3.3: Number of Pieces and Maximal Relative Error

3.4 Initialize with Feasible Point

When solving the scenario optimization on a middle-sized network, the Gurobi Optimizer may spend a huge amount of time in finding a feasible point. In most cases, providing a feasible initial point to the solver can significantly reduce the computational time because the Gurobi Optimizer will then be aware of an upper bound of the objective function. Therefore, we demonstrate a method to find a initial feasible point quickly.

First, set all tolls to be zero, i.e., $\boldsymbol{\tau} = \mathbf{0}$, and compute the equilibrium flows ${}_{eq}\boldsymbol{f}^{(i)}$ for all scenarios. Fixing $\boldsymbol{\tau}$ and ${}_{eq}\boldsymbol{f}^{(i)}$ in the reformulated optimization will make the bi-linear constraints linear. Thus we can efficiently solve the scenario optimization and obtain $\boldsymbol{\mu}^{(i)}$, $\boldsymbol{\lambda}^{(i)}$ and H. As a result, $(H, \boldsymbol{\tau} = \mathbf{0}, \boldsymbol{f}^{(i)} = {}_{eq}\boldsymbol{f}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\lambda}^{(i)})$ is a feasible point of the scenario optimization.

3.5 Support Subsample Evaluation

After obtaining the optimal solution for the scenario optimization, we are interested in the support subsample as declared in Definition 13, especially an irreducible one. Campi proposes a greedy algorithm to search for one support subsample in [11]. We adapt the algorithm to our scenario optimization in Algorithm 1.

Algorithm 1: Campi's Greedy Algorithm for Support Subsample

Result: An irreducible support subsample $\mathcal{S}_{irr}$
Set $\mathcal{S} \leftarrow \left(\delta^{(1)}, \ldots, \delta^{(N)}\right)$ and compute the optimal solution $\boldsymbol{\tau}^* \leftarrow \mathcal{A}_N(\mathcal{S})$;
for $i = 1, \ldots, N$ **do**
  Set $\bar{\mathcal{S}} \leftarrow \mathcal{S} \setminus \delta^{(i)}$ and compute the optimal solution $\bar{\boldsymbol{\tau}} \leftarrow \mathcal{A}_{|\bar{\mathcal{S}}|}(\bar{\mathcal{S}})$;
  if $\boldsymbol{\tau}^* = \bar{\boldsymbol{\tau}}$ **then**
    Set $\mathcal{S} \leftarrow \bar{\mathcal{S}}$;
  end
end
return $\mathcal{S}$

Algorithm 1 basically iterates all scenarios, removing those which do not contribute to the solution. In spite of the intuitiveness, the algorithm requires the scenario optimization to be solved for $N_S + 1$ times. Since each query may take quite a while, we propose a much faster greedy algorithm by exploiting the problem structure in Algorithm 2.

Algorithm 2: Greedy Algorithm to Construct Support Subsample

Result: A support subsample $\mathcal{S}_r$
Set $\mathcal{S} \leftarrow \emptyset$;
Compute the optimal solution $\boldsymbol{\tau}^* \leftarrow \mathcal{A}_N\left(\left(\delta^{(1)}, \ldots, \delta^{(N)}\right)\right)$;
Sort all scenarios in the descending order of the social cost given $\boldsymbol{\tau}^*$;
for $i = 1, \ldots, N$ **do**
  Set $\mathcal{S} \leftarrow \mathcal{S} \cup \delta^{(i)}$ and compute the optimal solution $\boldsymbol{\tau} \leftarrow \mathcal{A}_{|\mathcal{S}|}(\mathcal{S})$;
  if $\boldsymbol{\tau}^* = \boldsymbol{\tau}$ **then**
    Break;
  end
end
return $\mathcal{S}$

Algorithm 2 cumulatively collects scenarios that contribute to the solution. The scenarios are sorted in the descending order so that the one with higher social cost will be taken into account sooner. Note that the output $\mathcal{S}_r$ is not guaranteed irreducible but usually small enough. This algorithm involves $N_S + 1$ scenario optimizations *at most*. Unlike Algorithm 1 whose each optimization in the for-loop involves $N_S - 1$ scenarios, Algorithm 2 starts with 1 scenario and increases the number by one each time until N_S.

3.6 Experiments

3.6.1 Scenario Generation

For demonstrative purposes, we take the demands in [41] as the expectation and generate demand scenarios from different distributions, such as the uniform, Gaussian, and Poisson distribution, as are illustrated in Figure 3.4.

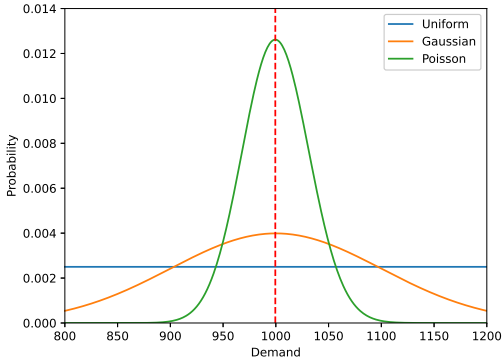


Figure 3.4: Different Distributions. The expectations are all 1000. The uniform distribution ranges from 800 to 1200; The Gaussian distribution has a standard deviation of 100.

Consider the Braess's network in Figure 2.3, we generate 365 scenarios from the uniform distribution, simulating the data collected through a year. The demands are ranged from 4.8 to 7.2, with 20% variations around the expected value 6. Figure 3.5 illustrates these sampled demands.

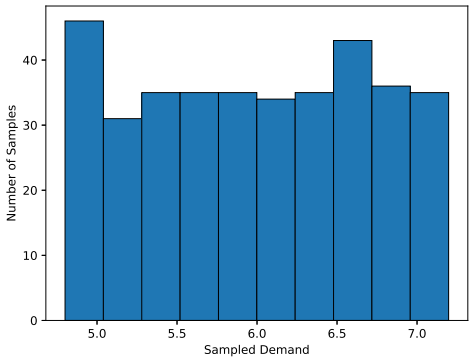


Figure 3.5: Sampled Demands from Uniform Distribution

3.6.2 Result Analysis

The scenario optimization gives the probabilistically-robust optimal tolls $\boldsymbol{\tau}^* = [11, 0, 0, 0, 11]$ and the optimal worst-case social cost $H^* = 644$. The cardinality of the support subsample found by Algorithm 1 is 1, implying the support subsample is actually irreducible. Per Therorem 14, we conclude: the violation probability of $H \leq H^*$ is no larger than 0.068 with the confidence of $1 - 10^{-6}$. To verify the argument, we draw another $10,000$ random scenarios and compute the social costs given $\boldsymbol{\tau}^*$.

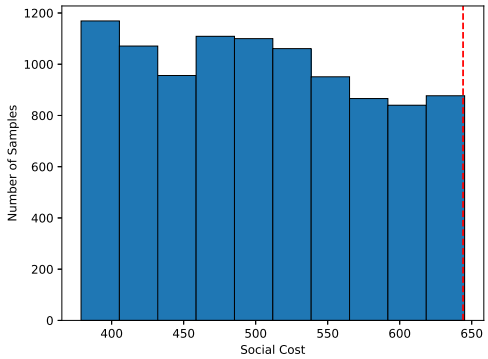


Figure 3.6: Social Cost Histogram

We present the histogram of the social costs in Figure 3.6 and highlight H^* with a red dashed line. There are 32 scenarios causing a social cost larger than 644, i.e., $\mathbb{P}\{H > H^*\} = 0.0032$. We repeated the experiment 100 times and $\mathbb{P}\{H > H^*\} < 0.068$ is always true. Such behavior is as expected because Theorem 14 guarantees that $\mathbb{P}\{H > H^*\} < 0.068$ holds true with the confidence of $1 - 10^{-6}$ - almost unlikely to violate.

3.6.3 Discussion

It is well-known that the inefficiency of the Braess's network can be completely eliminated by removing the middle link 4, or equivalently imposing an infinite toll on it. The scenario optimization, however, found another solution leading to the same optimal objective due to the intrinsic non-convexity. Besides the unlimited tolls, we can also easily restrict the tolls under some upper bounds. For example, if we allow $\tau_e = 5$ at most, the corresponding robust optimal tolls will be $\boldsymbol{\tau} = [4.7, 0, 0, 5, 4.7]$.

Thanks to the simple structure of the Braess's network, we can analytically compute the *minimal* optimal tolls. To compare with the previous solution, we impose tolls only on the link 1 and 5. When the user equilibrium coincides with the social optimum, users should have no incentive to alter from the social optimum. Per Definition 5, the cost accumulated along any path should be less than or equal to the paths not used. At the socially optimal flows, half of players choose the upper way and the others choose the lower way, with nobody passing through the link 4, i.e., $f_1 = f_2 = f_3 = f_5 = 0.5d, f_4 = 0$. Symmetrically, we impose τ on both the link 1 and 5. The

tolls enforcing the socially optimal flow should follow the inequality.

$$\begin{aligned} l_1(f_1) + \tau + l_3(f_3) &\leq l_1(f_1) + \tau + l_4(f_4) + l_5(f_5) + \tau \\ l_3(f_3) &\leq l_4(f_4) + l_5(f_5) + \tau \\ f_3 + 50 &\leq f_4 + 10 + 10f_5 + \tau \\ 0.5d + 50 &\leq 0 + 10 + 5d + \tau \\ \tau^* &\geq 40 - 4.5d \end{aligned}$$

When the demand is at minimum 4.8, the minimal optimal toll is 18.4, while at maximum 7.2, the toll is 7.6. The solution returned by the scenario approach is 11, exactly within $[7.2, 18.4]$, which means the obtained solution cannot always force the socially-optimal flow. In fact, it was not even intended to robustly incentivize the socially-optimal flow because the goal was to robustly minimize the worst-case social cost. One who is interested in the inefficiency mitigation should resort to (SA-P) instead of (SA-H). The reformulations are very similar to the ones introduced in this chapter and thus are omitted.

If we explicitly state that only the middle link is taxable, the scenario approach will give intuitive tolls $\boldsymbol{\tau} = [0, 0, 0, 13, 0]$. Once again, we can compute the minimal tolls analytically as follows.

$$\begin{aligned} l_1(f_1) + l_3(f_3) &\leq l_1(f_1) + l_4(f_4) + \tau + l_5(f_5) \\ l_3(f_3) &\leq l_4(f_4) + \tau + l_5(f_5) \\ f_3 + 50 &\leq f_4 + 10 + \tau + 10f_5 \\ 0.5d + 50 &\leq 0 + 10 + \tau + 5d \\ \tau^* &\geq 40 - 4.5d \end{aligned}$$

Interestingly, the derivation shows that imposing a toll on the middle link is equivalent to imposing the same toll on both the link 1 and the link 5.

As for the computational cost, it takes as much as a whole day to solve the scenario approach with 365 scenarios for the extremely simple Braess's network. The fact discourages any further application in the real traffic systems. The formidable computational cost for the global optimum motivates us to turn to a decent suboptimal solution instead. Therefore, in the next chapter, we will propose several algorithms to trade the performance for the efficiency.

Chapter 4

Algorithms for Suboptimal Solution

4.1 Greedy Numerical Gradient Descent Algorithm

4.1.1 Model Reformulation

Let us take a second look at (SA-P). Note that $\boldsymbol{f}_{so}^{(i)}$ does not depend on $\boldsymbol{\tau}$ or P, which enables us to pre-compute $\boldsymbol{f}_{so}^{(i)}$ before starting the scenario optimization. For the ease of notation, we denote the corresponding social cost as a constant ${C_{soc}^*}^{(i)}$. Further, define $P^{(i)}$ to be the price of anarchy of the i^{th} scenario. Reformulate (SA-P) as follows.

$$
\begin{aligned}
&\min\ P && \text{(SA-P-2)}\\
&\text{subject to:}\\
&\qquad P = \max P^{(i)}\\
&\qquad P^{(i)} = \frac{\sum_e f_{ue,e}^{(i)} l_e(f_{ue,e}^{(i)})}{{C_{soc}^*}^{(i)}}\\
&\qquad \boldsymbol{f}_{ue}^{(i)} = \arg\min \sum_e \int_0^{f_{ue,e}^{(i)}} l_e(t) + \tau_e \,\mathrm{d}t\\
&\qquad\quad \boldsymbol{A}\boldsymbol{f}_{ue}^{(i)} = \boldsymbol{b}^{(i)}\\
&\qquad\quad \boldsymbol{f}_{ue}^{(i)} \geq \boldsymbol{0}
\end{aligned}
$$

Given $\boldsymbol{\tau}$, we can efficiently compute $\boldsymbol{f}_{ue}^{(i)}$ with many traffic assignment algorithms, such as the Frank-Wolfe algorithm, and once we have $\boldsymbol{f}_{ue}^{(i)}$, we can directly obtain $P^{(i)}$ as well as P. In other words, the objective P can be regards as a function of $\boldsymbol{\tau}$ solely. The fact motivates us to propose a numerical gradient descent algorithm to reduce the worst-case PoA iteratively.

4.1.2 Frank-Wolfe Algorithm

The Frank-Wolfe algorithm is an iterative first-order optimization algorithm for the constrained convex optimization [19], well suited for solving (OPT-UE). The algorithm is stated as follows. Suppose $\mathcal{D}$ is a compact convex set in a vector space and $f\colon \mathcal{D} \to \mathbb{R}$ is a convex, differentiable real-valued function. The Frank-Wolfe algorithm solves the following optimization problem

$$
\begin{aligned}
&\min_{\boldsymbol{x}}\ f(\boldsymbol{x}) && \text{(FW)}\\
&\text{subject to: } \boldsymbol{x} \in \mathcal{D}.
\end{aligned}
$$

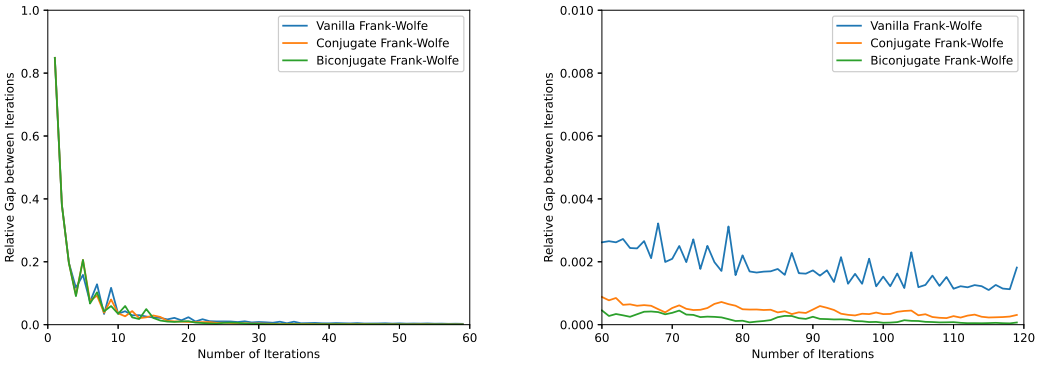
The algorithm is formalized in Algorithm 3.

Algorithm 3: Frank-Wolfe Algorithm

Result: The optimal solution $\boldsymbol{x}^*$ to (FW)
Let $k \leftarrow 0$ and let $\boldsymbol{x}_0$ be any point in $\mathcal{D}$;
Step 1. Direction-finding subproblem:
Solve $\boldsymbol{s}_k \leftarrow \arg\min_{\boldsymbol{s}\in\mathcal{D}} \ \boldsymbol{s}^\top \nabla f(\boldsymbol{x}_k)$;
Step 2. Step size determination:
Set $\alpha \leftarrow \frac{2}{k+2}$ or line search for $\alpha \leftarrow \arg\min_{\alpha\in[0,1]} f(\boldsymbol{x}_k + \alpha(\boldsymbol{s}_k - \boldsymbol{x}_k))$;
Step 3. Update:
Let $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \alpha(\boldsymbol{s}_k - \boldsymbol{x}_k)$, let $k \leftarrow k+1$ and go to Step 1;
return $\boldsymbol{x}_k$

In the initialization step, $\boldsymbol{x}_0$ can be any feasible traffic flow. Step 1 simply corresponds to solve the shortest path for each commodity and assign the whole demand of this commodity along the shortest path (all-or-nothing link flows). The Dijkstra's shortest path algorithm can tackle Step 1 efficiently. Step 1-3 forms a complete iteration, and we usually terminate the algorithm after a pre-set number of iterations or when the objective value varies within a super small threshold.

There are two widely-used variants of the vanilla Frank-Wolfe algorithm (VFW), namely, conjugate Frank-Wolfe algorithm (CFW) and bi-conjugate Frank-Wolfe algorithm (BFW). Take the Sioux Falls network for example, we demonstrate in Figure 4.1 how the relative objective gap of (OPT-UE) between two iterations varies with respect to the number of iterations.



(a) Early Stage (b) Late Stage

Figure 4.1: Convergence of Different Frank-Wolfe Algorithms

Although these three algorithms all quickly reduce the relative gap in the early stage, the biconjugate one surpasses other two in the late stage, in terms of smoothness and performance. To quantitatively compare their performance, we set different thresholds for the relative gap and record the iterations and time they take to converge, in Table 4.1.

Table 4.1: Iteration and Time before Convergence of Different Frank-Wolfe Algorithms. The algorithms are evaluated with different convergence thresholds (TH). The numbers of iterations and the time before convergence are specified while N/A means the algorithm did not converge within 1000 iterations.

Algorithm	**TH 1e-2**		**TH 1e-3**		**TH 1e-4**		**TH 1e-5**	
	#Iter	Time	#Iter	Time	#Iter	Time	#Iter	Time
VFW	30	0.249s	132	0.576s	N/A	N/A	N/A	N/A
CFW	21	0.311s	56	0.418s	204	0.974s	N/A	N/A
BFW	19	0.270s	44	0.380s	84	0.523s	301	1.365s

As a result, we choose the biconjugate Frank-Wolfe algorithm as the traffic assignment algorithm because it excels at both the performance and efficiency. However, we should be aware that in spite of the high precision, the algorithm will not converge to an exact value due to the numerical error, which will be passed to and augmented by later calculations.

4.1.3 Greedy Numerical Gradient Descent Algorithm

The scalar-valued function P is generally non-differentiable in $\boldsymbol{\tau}$, that is, the gradient is not well-defined everywhere. Therefore, in this chapter we talk about the gradient in a generalized sense. We will approximate the gradient by the *numerical* partial derivatives. At points where P is non-differentiable or even discontinuous, the numerical gradient does not make sense and the gradient descent may even worsen the objective. Fortunately, our numerical gradient descent algorithm will not be invalidated given proper adaptions as shown later. Per gradient definition,

$$\begin{aligned}\nabla P(\boldsymbol{\tau}) = \begin{bmatrix} \frac{\partial P}{\partial \tau_1} \\ \vdots \\ \frac{\partial P}{\partial \tau_{N_E}} \end{bmatrix} &= \lim_{\delta\tau \to 0} \begin{bmatrix} \frac{P(\tau_1+\delta\tau,\boldsymbol{\tau}_{-1})-P(\tau_1-\delta\tau,\boldsymbol{\tau}_{-1})}{2\delta\tau} \\ \vdots \\ \frac{P(\tau_{N_E}+\delta\tau,\boldsymbol{\tau}_{-N_E})-P(\tau_{N_E}-\delta\tau,\boldsymbol{\tau}_{-N_E})}{2\delta\tau} \end{bmatrix} \\ &\approx \begin{bmatrix} \frac{P(\tau_1+\delta\tau,\boldsymbol{\tau}_{-1})-P(\tau_1-\delta\tau,\boldsymbol{\tau}_{-1})}{2\delta\tau} \\ \vdots \\ \frac{P(\tau_{N_E}+\delta\tau,\boldsymbol{\tau}_{-N_E})-P(\tau N_E-\delta\tau,\boldsymbol{\tau}_{-N_E})}{2\delta\tau} \end{bmatrix}, \end{aligned} \tag{4.1}$$

where $\boldsymbol{\tau}_{-i}$ refers to $\tau_1, \ldots, \tau_{i-1}, \tau_{i+1}, \ldots, \tau_{N_E}$, i.e., all other tolls except τ_i. In most cases, there are upper bounds and lower bounds (usually 0) for tolls so we modify (4.1) to respect the bounds.

$$\nabla P(\boldsymbol{\tau}) \approx \begin{bmatrix} \frac{P(\tau_1+\min(\delta\tau,\tau_1^+-\tau_1),\boldsymbol{\tau}_{-1})-P(\tau_1-\min(\delta\tau,\tau_1-\tau_1^-),\boldsymbol{\tau}_{-1})}{\min(\delta\tau,\tau_1^+-\tau_1)+\min(\delta\tau,\tau_1-\tau_1^-)} \\ \vdots \\ \frac{P(\tau_{N_E}+\min(\delta\tau,\tau_{N_E}^+-\tau_{N_E}),\boldsymbol{\tau}_{-N_E})-P(\tau_{N_E}-\min(\delta\tau,\tau_{N_E}-\tau_{N_E}^-),\boldsymbol{\tau}_{-N_E})}{\min(\delta\tau,\tau_{N_E}^+-\tau_{N_E})+\min(\delta\tau,\tau_{N_E}-\tau_{N_E}^-)} \end{bmatrix}, \tag{4.2}$$

where τ_i^+ and τ_i^- mean the upper bound and lower bound of the toll on link i. (4.2) means that if the function augment is outside the feasible region, we project the augment onto the bound and adapt the gradient calculation correspondingly.

We should be aware that each gradient (4.2) involves $2M_\tau N_s$ traffic assignments, where M_τ is the number of taxable links. Although one traffic assignment can be solved efficiently, $2M_\tau N_s$ is usually a huge number. For example, consider the Sioux Falls network with 76 edges all taxable. If we collect 365 samples, each gradient involves $27,740$ traffic assignments. However, most of the scenarios have no effect on the solution and the support subsamples. Therefore, we propose a greedy algorithm to lazily take scenarios into account.

Algorithm 4: Greedy Numerical Gradient Descent

Input: maximal iteration K_{max}, convergence iteration K_{conv}, initial tolls $\boldsymbol{\tau}_0$, delta tau $\delta\tau$, step size decision function $\gamma(\cdot)$, convergence threshold θ;
Result: Suboptimal tolls $\boldsymbol{\tau}$ and support subsample $\mathcal{S}$;
$k \leftarrow 0, k_{conv} \leftarrow 0$;
$i \leftarrow \arg\max_{i\in\{1,\dots,N_s\}} P^{(i)}$;
$\mathcal{S}_0 \leftarrow \{i\}$;
while $k < K_{max}$ **do**
  while *True* **do**
    $\nabla\bar{P}_k \leftarrow$ (4.2) with $\delta\tau, \boldsymbol{\tau}_k, \mathcal{S}_k$;
    $\bar{\boldsymbol{\tau}}_{k+1} \leftarrow \boldsymbol{\tau}_k - \gamma(k)\nabla\bar{P}_k$;
    $\bar{i} \leftarrow \arg\max_{i\in\{1,\dots,N_s\}} P^{(i)}(\bar{\boldsymbol{\tau}}_{k+1})$;
    if $\bar{i} \in \mathcal{S}_k$ **then**
      $\boldsymbol{\tau}_{k+1} \leftarrow \bar{\boldsymbol{\tau}}_{k+1}$;
      $P_{k+1} \leftarrow P(\boldsymbol{\tau}_{k+1})$;
      Break
    else
      $\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{\bar{i}\}$
    end
  end
  $\mathcal{S}_{k+1} \leftarrow \mathcal{S}_k$;
  $k \leftarrow k + 1$;
  if $|P_{k+1} - P_k| < \theta$ **then**
    $k_{conv} \leftarrow k_{conv} + 1$;
    if $k_{conv} = K_{conv}$ **then**
      Break
    end
  else
    $k_{conv} \leftarrow 0$
  end
end
Return $\boldsymbol{\tau}_k, \mathcal{S}_k$

The key idea of Algorithm 4 is that, at each iteration, we compute the numerical gradient $\nabla\bar{P}_k$ only with respect to the current support subsample set $\mathcal{S}_k$. We try a tentative toll $\bar{\boldsymbol{\tau}}_{k+1}$ from the numerical gradient descent and verify whether the support subsample set remains the same. If not, include the one that violates the most into the support subsample set and repeat the verification. The variable k_{conv} means the number of consecutive iterations that the difference between the previous and current P is within a threshold θ. If the difference of PoA between two iterations stays within the threshold θ for a given number of consecutive iterations K_{conv}, we say the gradient descent algorithm *converges*. As a result, $\mathcal{S}_k$ is a support subsample set, and thus we obtain the cardinality of the support subsample for free.

4.1.4 Delta Tau

The choice of $\delta\tau$ requires careful consideration. A large $\delta\tau$ cannot well approximate the partial derivative, while a small $\delta\tau$ causes virtually invisible change in the real social cost - however, the social cost is computed by the numerical algorithm and is susceptible to the numerical error. It is highly possible that the numerical error is much greater than the real social cost change. Figure 4.2 shows how the price of anarchy varies during the gradient descent iterations.

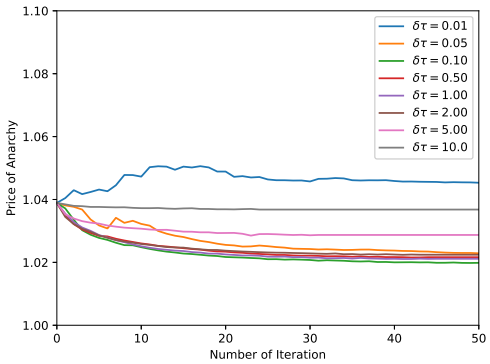


Figure 4.2: Different Delta Tau. The experiment was conducted in the Sioux Falls network with the nominal demands, all parameters kept consistent except for $\delta\tau$.

As a result, $\delta\tau = 0.01$ gives the worst result while $\delta\tau = 0.10$ performs best.

4.1.5 Toll Post-Process

Toll Rounding: The gradient descent algorithm is likely to return tolls with a large decimal place. We would like to reduce the decimal place in order to improve the numerical stability. Moreover, in reality tolls are expected to be regular number with a small decimal place. Figure 4.3 shows the price of anarchy curve with respect to the iteration number given different rounding schemes.

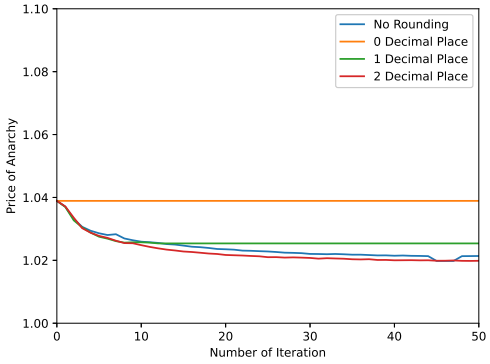


Figure 4.3: Toll Rounding

When tolls are rounded to integers, the price of anarchy curve is a horizontal line because the step size was not able to change the toll value more than 1, and the tolls stay the same after rounding. Afterwards, we will round all tolls to two decimal places.

Toll Projection: Since the gradient descent algorithm above does not explicitly deal with the bounding constraints of tolls, we need to project tolls back to the feasible region to respect bounds. One simple method is element-wise projecting infeasible tolls onto their bounds.

4.1.6 Cache for PoA

In the second while loop of Algorithm 4, we may compute $\nabla \bar{P}_k$ multiple times. Each computation requires the traffic assignment $2M_\tau|\mathcal{S}_k|$ times. However, we only need to assign the traffic for the newly-introduced scenario into the support subsample. For example, assume at some iteration k, the current subsample set is $\mathcal{S}_k = \{1, 2\}$ and after the verification, we decide that we have to add scenario 3 to the support subsample set. To compute $\nabla \bar{P}_k$, instead of all $6M_\tau$ traffic assignments, we only need to assign $2M_\tau$ times for the newly-included scenario 3, because the $4M_\tau$ traffic assignments have already been finished when we get $\mathcal{S}_k = \{1, 2\}$. Therefore, we utilize a cache to temporarily store $P^{(i)}$ to avoid repeated calculation. The cache should be reset after each iteration.

4.1.7 Step Size Determination

We adopt a piece-wise function to determine the step size γ,

$$\gamma(k) = \begin{cases} \frac{500}{k} & k \leq 20 \\ \frac{25000}{k^2} & k > 20 \end{cases}, \tag{4.3}$$

where k is the number of iterations. We use a common step size choice for the first piece, and accelerate the diminishing by a quadratic denominator k^2 in the second piece. The numerators are chosen to make γ continuous at $k = 20$.

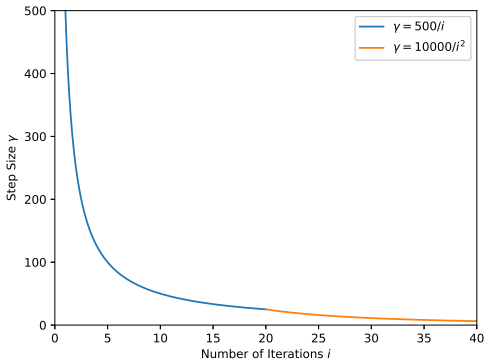


Figure 4.4: Step Size γ

The numerical derivative computed with (4.2) may be very large and cause the objective value to fluctuate a lot. To mitigate the unstable fluctuation, we force the tolls to vary only within a small range, by normalizing them when the infinity norm exceeds a certain threshold, for example, 1,

$$\boldsymbol{\tau}_{k+1} = \boldsymbol{\tau}_k - \frac{\gamma_k \nabla P(\boldsymbol{\tau}_k)}{\max\left(1, \|\boldsymbol{\tau}_k\|_\infty\right)}. \tag{4.4}$$

Because the objective function is somewhere non-differential and even discontinuous, the descent along the generalized numerical gradient may worse the objective as seen in Figure 4.2. To suppress the backfire, we propose an algorithm to greedily search for a *desirable* step size. The key idea to progressively explore toll candidates that are most likely to reduce the objective

function. It also makes use of a set OPEN, which is a current list of nodes that could potentially be a desirable one. Let $O(\cdot)$ denote the objective function to *minimize*, and tuple $(\boldsymbol{\tau}_l, \boldsymbol{\tau}_r)$ denote a node in the OPEN set. OPEN is a priority queue, and nodes with lower $O(\boldsymbol{\tau}_l) + O(\boldsymbol{\tau}_r)$ are associated with higher priority. Finally, let $\boldsymbol{\tau}_k$ refer to the toll in the previous iteration, and $\boldsymbol{\tau}_{k+1}$ is the toll computed by the gradient descent (4.4). The algorithm is formulated as follows.

Algorithm 5: Greedy Binary Search for Step Size

Input: Previous toll $\boldsymbol{\tau}_k$, toll from gradient descent $\boldsymbol{\tau}_{k+1}$, maximal iterations i_{max};
Result: Desirable toll $\boldsymbol{\tau}$ such that $O(\boldsymbol{\tau}) \leq O(\boldsymbol{\tau}_k)$;
if $O(\boldsymbol{\tau}_k) > O(\boldsymbol{\tau}_{k+1})$ **then**
  return $\boldsymbol{\tau}_{k+1}$
end
Add $(\boldsymbol{\tau}_k, \boldsymbol{\tau}_{k+1})$ to OPEN;
while $i < i_{max}$ **do**
  $(\boldsymbol{\tau}_l, \boldsymbol{\tau}_r) \leftarrow$ the first node in OPEN;
  Remove the first node in OPEN;
  $\boldsymbol{\tau} \leftarrow (\boldsymbol{\tau}_l + \boldsymbol{\tau}_r)/2$;
  if $O(\boldsymbol{\tau}) < O(\boldsymbol{\tau}_k)$ **then**
    return $\boldsymbol{\tau}$
  end
  Add $(\boldsymbol{\tau}_l, \boldsymbol{\tau})$ to OPEN;
  Add $(\boldsymbol{\tau}, \boldsymbol{\tau}_r)$ to OPEN;
  $i \leftarrow i + 1$
end
Return $\boldsymbol{\tau}_k$ and terminate gradient descent

If the algorithm cannot find a desirable toll within i_{max} iterations, the gradient descent will terminate and take $\boldsymbol{\tau}_k$ as the final toll. We illustrate the algorithm with an simple example, where the objective function only takes one scalar-valued parameter. Figure 4.5 shows the objective function O against the toll τ. At some iteration k, $\tau_k = 0$ and $\tau_{k+1} = 1$. The algorithm first verifies $O(\tau_{k+1}) > O(\tau)$ and enters the while loop. Then by progressively explore 0.5 and 0.25, the algorithm finally finds a desirable toll $\tau = 0.125$ and returns it. We also highlight that the best tolls at 0.8 cannot be found by this algorithm.

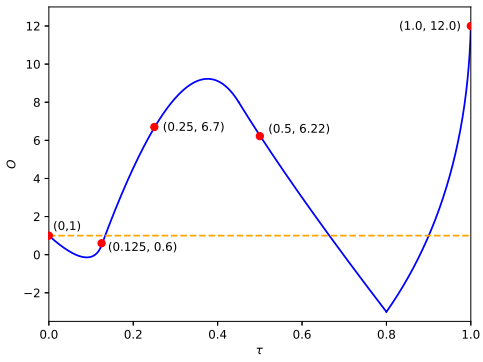
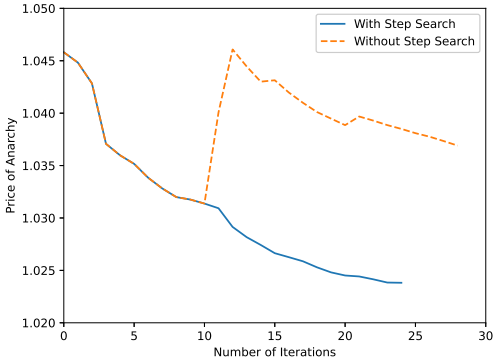
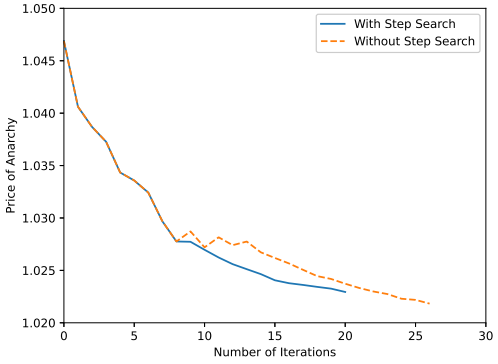


Figure 4.5: Greedy Binary Search for Step

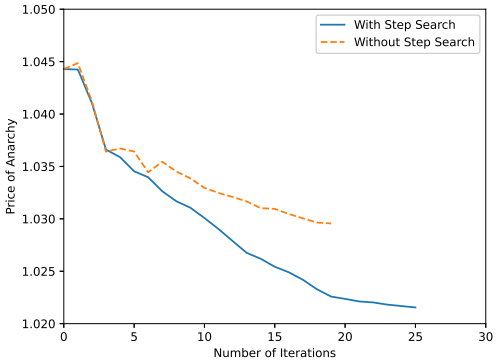
If the price of anarchy curve resulting from the gradient descent algorithm is already monotonically decreasing until convergence without Algorithm 4.5, the step search will not take any effect. Otherwise, Figure 4.6 highlights possible outcomes regarding the efficiency (number of iterations before convergence) and performance (the final price of anarchy).



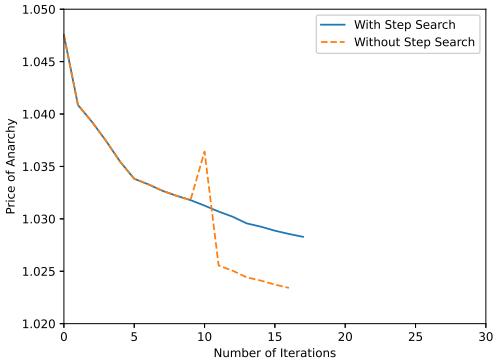
(a) Less Iterations Better Performance



(b) Less Iterations Worse Performance



(c) More Iterations Better Performance

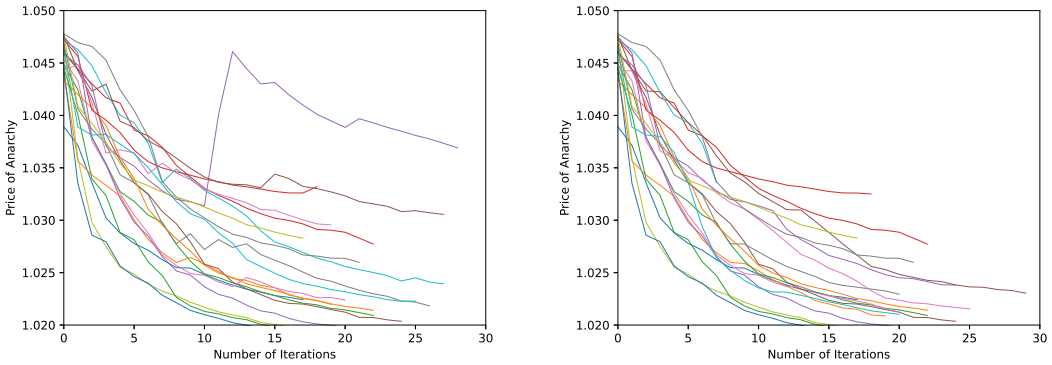


(d) More Iterations Worse Performance

Figure 4.6: With and Without Greedy Binary Search for Step Size

4.1.8 Multi-Start Strategy

The greedy binary search algorithm for a desirable step size may terminate early and reduce the price of anarchy only slightly. A natural idea to improve the performance is starting the gradient descent algorithm from multiple initial tolls. The descending tracks of different initials are independent and thus can be processed in parallel. Figure 4.7 exemplifies the multi-start PoA of the Sioux Falls network with the nominal demands. The initial tolls are generated from the uniform distribution on $[0, 1]$. We then started the gradient descent without (Figure 4.7a) and with (Figure 4.7b) the step size search algorithm. Note that the initial toll samples are the same for these two cases.



(a) Without Step Search

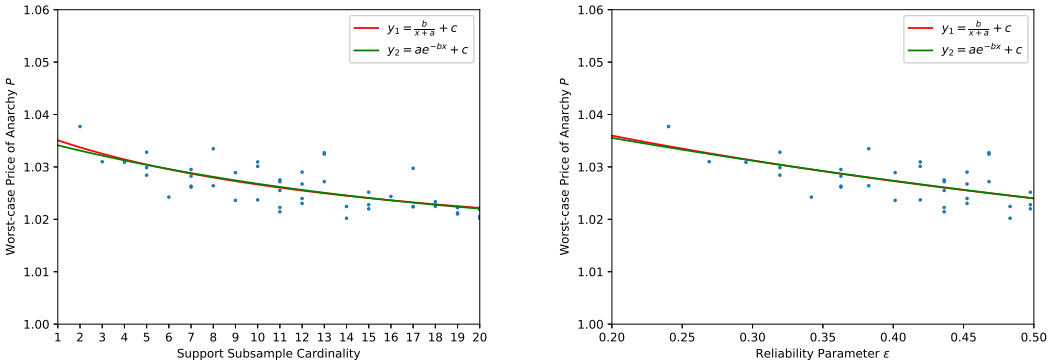
(b) With Step Search

Figure 4.7: Multi-start Gradient Descent

Besides the better performance, the multi-start strategy also provides a richer result. All the initial points go through a complete numerical gradient descent process, and thus we can take them as independent experiments. Each experiment returns a worse-case price of anarchy and a support subsample set. In other words, these multiple initial points also bring multiple performance-robustness pairs. Some initial points may result in a better performance at the cost of a large support subsample set, and others in the other way around. The decision-maker can prioritize the performance or the robustness based on the preference.

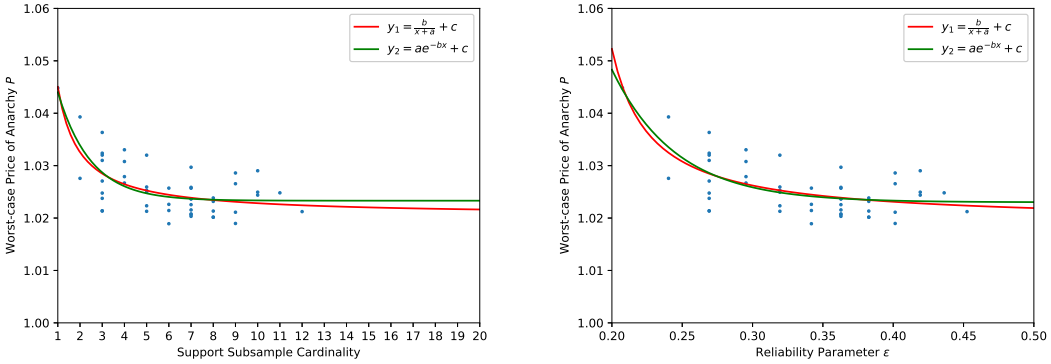
4.1.9 Result Analysis

With all the modifications, we finally present results on the Sioux Falls network. Assume all links are taxable. We generate 100 demand scenarios from the uniform distribution with different variations. The variation means how much at most the demand scenario can deviate from the nominal demands. For example, with 5% variation and the nominal demand of 100, the demand scenario is a random variable uniformly distributed in $[100(1-5\%), 100(1+5\%)]$. Figure 4.8-4.12 report the performance and robustness under different variation settings with 50 initial points. When the numerical gradient descent algorithm terminates, each initial point corresponds to a performance-robustness pair, presenting as a point in the figure. The performance is evaluated based on the worse-case price of anarchy while the robustness is quantified by the reliability parameter, which can be inferred from the cardinality of the support subsample set. We also fit the curves with a generalized inverse proportional function and an exponential function.



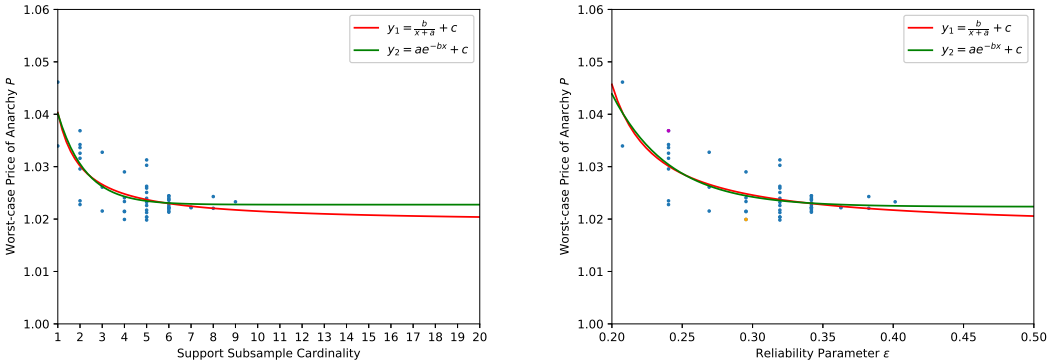
(a) PoA and Support Subsample (b) PoA and Reliability

Figure 4.8: Performance and Robustness (Variation: 1%)



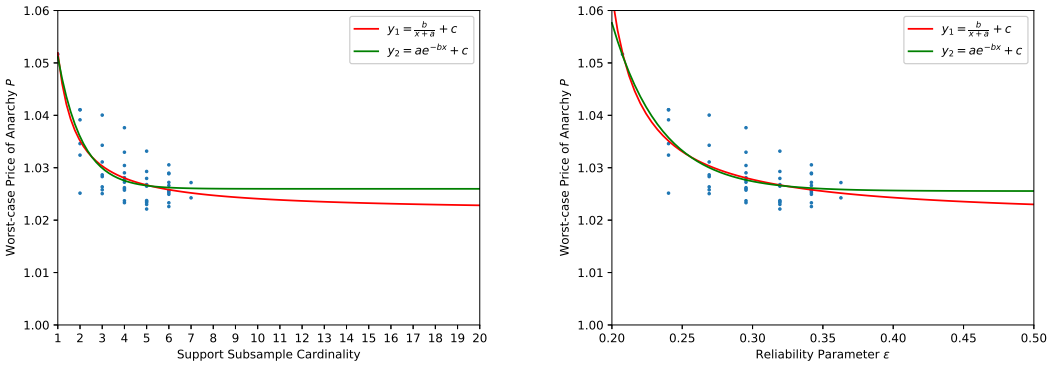
(a) PoA and Support Subsample (b) PoA and Reliability

Figure 4.9: Performance and Robustness (Variation: 2%)



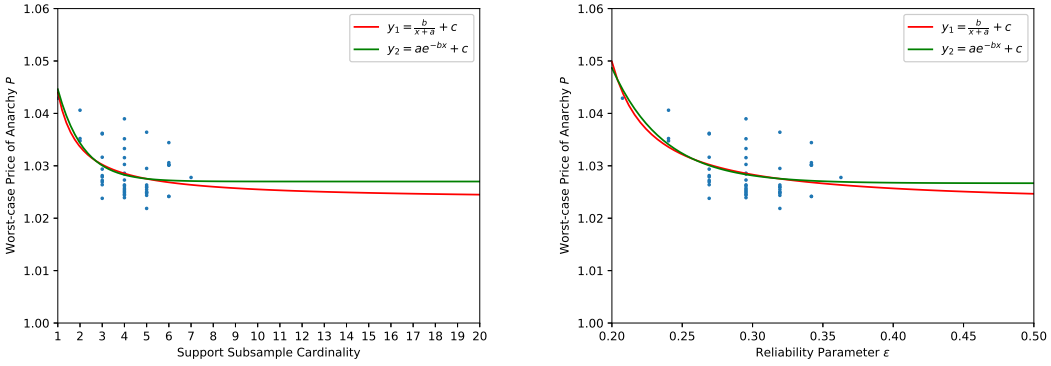
(a) PoA and Support Subsample (b) PoA and Reliability

Figure 4.10: Performance and Robustness (Variation: 5%)



(a) PoA and Support Subsample (b) PoA and Reliability

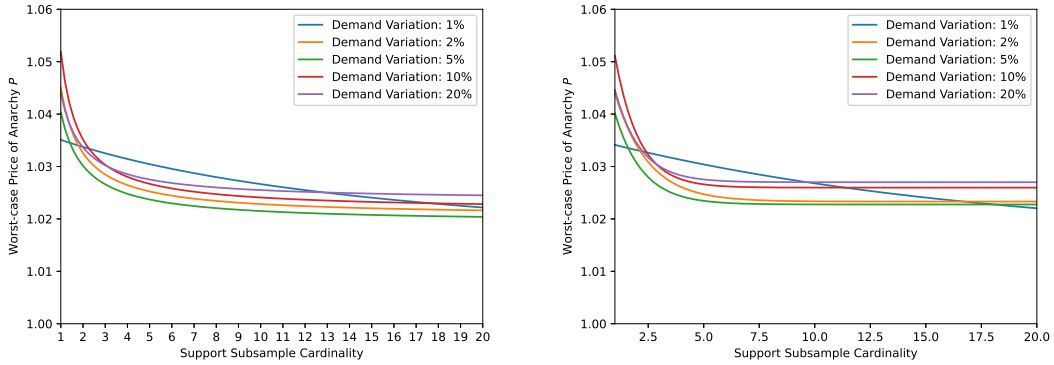
Figure 4.11: Performance and Robustness (Variation: 10%)



(a) PoA and Support Subsample (b) PoA and Reliability

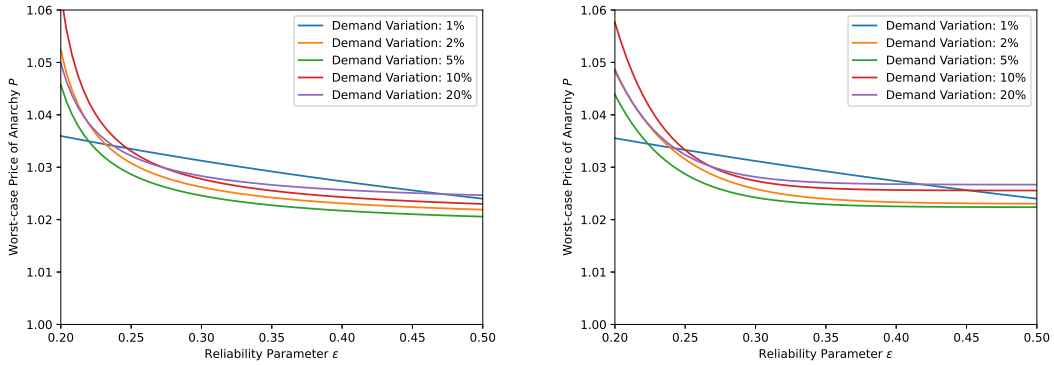
Figure 4.12: Performance and Robustness (Variation: 20%)

Let Figure 4.13 and Figure 4.14 collect all these fitted curves.



(a) Generalized Inverse Proportional Function (b) Exponential Function

Figure 4.13: PoA and Support Subsample Curve Fitting



(a) Generalized Inverse Proportional Function (b) Exponential Function

Figure 4.14: PoA and Reliability Parameter Curve Fitting

As the maximal variation increases, more and more points are scattered in the left part of the figure, which can be interpreted that more variation leads to more robustness. This may seem counter-intuitive at first, but if we pause to consider how the algorithm actually operates, this fact becomes obvious - small variations mean the PoA of different scenarios are close to each other, making the condition $\bar{i} \in \mathcal{S}_k$ in Algorithm 4 less likely to happen, so the algorithm will collect more scenarios into the support subsample set, leading to less robustness.

4.1.10 Toll Performance Comparison

In this section, we compare the performance of the margin-cost tolls and probabilistically robust tolls from the scenario approach. We exemplify the Sioux Falls network with 5% demand variation. As stated in Section 4.1.8, the multi-start strategy provides multiple toll sets with different performance-robustness properties. We take two sets of tolls for instance. Let Toll1 and Toll2 refer to the purple point $(0.240, 1.037)$ and the orange point $(0.295, 1.020)$ in Figure 4.10b, respectively. Toll1 is selected for its better robustness (second smallest reliability parameter ϵ), and Toll2 stands out because of the excellent balance between performance and robustness.

We start the comparison by plotting the toll values on the 76 edges in Figure 4.15. We can observe that the marginal-cost tolls have large magnitudes and fluctuations while the largest toll value from the scenario approach is only 2.21.

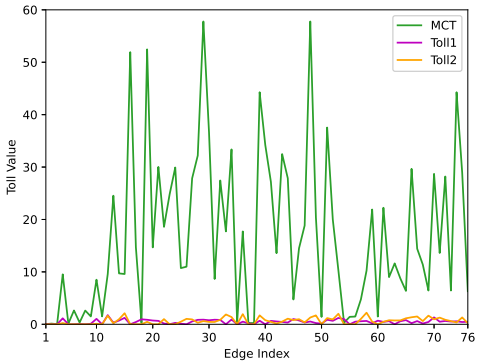


Figure 4.15: Toll Value Comparison

To compare the performance and robustness, we generate 36, 500 new scenarios with 5% variation, and compute the social cost with these different tolls. To better illustrate the performance, we also include the social cost of Wardrop equilibrium flow without any tolls.

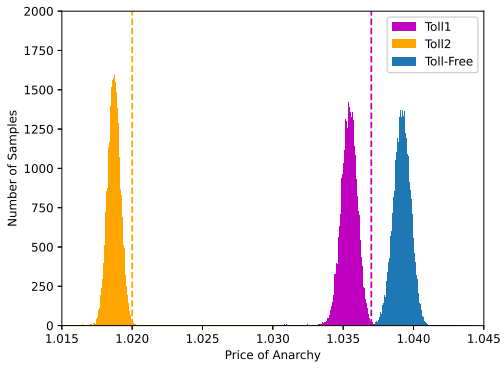


Figure 4.16: Toll Performance Comparison (Variation: 5%)

Figure 4.16 shows the histogram of PoA. The vertical dashed lines are at 1.037 and 1.020. When we charge Toll1, there are 143 (0.39%) scenarios whose PoA is greater than 1.037, while given Toll2, 121 (0.33%) scenarios perform worse than PoA 1.020. Both Toll1 and Toll2 improve the network efficiency effectively, compared to the toll-free PoA, illustrated with the blue bars.

However, we did not include the performance of marginal-cost tolls in Figure 4.16 because they almost incentivize the optimal traffic flows, and most of scenarios accumulate in a single bar at PoA 1.00. Although the marginal-cost tolls are robust in this experiment, they achieve the seemly good performance in an unwanted way - large toll values almost prohibit everyone from using some links, and the links are taxed extremely unevenly, as shown in Figure 4.15.

We continue to compare the performance on the Sioux Falls network with 20% variation in Figure 4.17. We can notice that the three areas are significantly "fatter" because PoAs of the scenarios are more different with more variation. In this experiment, Toll1 corresponding to $(0.240, 1.034)$ results in 654 (1.79%) scenarios with a larger PoA than 1.034, and Toll2 corresponding to $(0.295, 1.024)$ results in 617 (1.69%) scenarios with a larger PoA than 1.024.

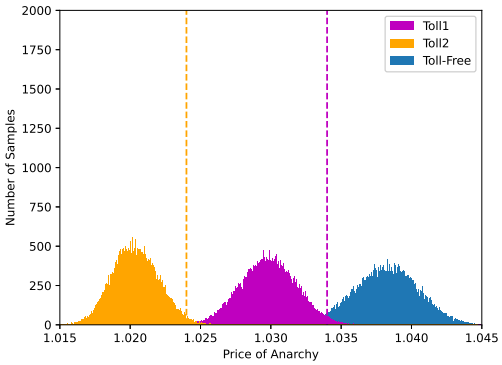
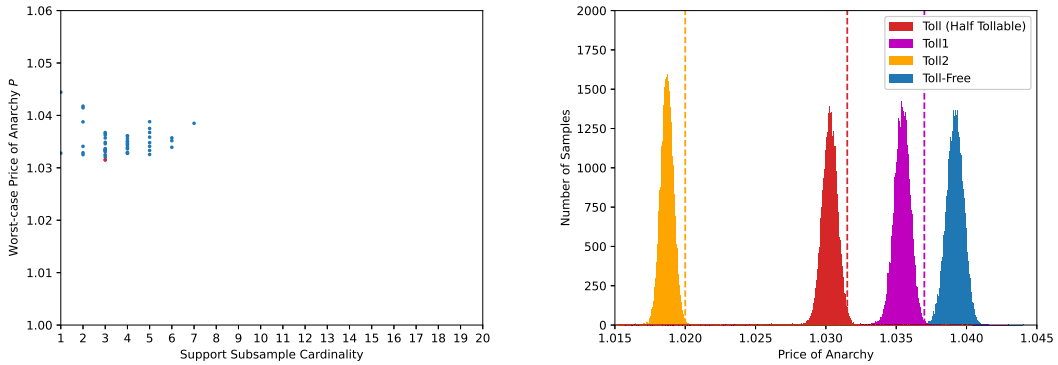


Figure 4.17: Toll Performance Comparison (Variation: 20%)

In conclusion, the numerical gradient descent algorithm can find multiple sets of tolls, which achieve desirable performance and guarantee probabilistic robustness.

4.1.11 Performance on Partially-Taxable Networks

Finally, we consider the partially-taxable networks. Assume that only half of roads in the Sioux Falls network can be charged. We randomly pick 38 roads from all 76 roads and run the greedy numerical gradient descent algorithm with 50 initial points. The relation between the price of anarchy and the support subsample cardinality is shown in Figure 4.18a. We pick the toll set with the best worst-case price of anarchy (highlighted in red) and impose them on the network. We run the experiment with 36,500 new scenarios and illustrate the resulting social costs in Figure 4.18b. Note that the only difference between Figure 4.16 and Figure 4.18b is that the latter includes the red area.



(a) PoA and Support Subsample

(b) Toll Performance Comparison

Figure 4.18: Performance and Robustness (Variation: 5%, Half-taxable)

We can observe that the red toll set performs worse than the orange one. Intuitively, the more roads we can tax, the more control we have of the network. When we are only able to tax half roads, the best performance naturally declines. However, the best performance is not simply a function of the number of taxable roads since each road contributes differently to the inefficiency. In the case that we only want to tax a certain number of roads, it is hard to choose the taxable roads because we cannot easily evaluate their potential inefficiency.

Figure 4.18b also shows that the red toll set performs better than the magenta one, since the PoA of the red one (1.037) is smaller than the PoA of the magenta one (1.032). On the other hand, the number of the scenarios with a higher PoA of the red dash vertical line is 423, much more than that of the magenta one - 79. The fact reminds us that we should never discuss performance solely without robustness, or vice versa, because the performance and robustness always come as a bounded pair.

In conclusion, the numerical gradient descent algorithm also applies to partially-taxable networks.

4.2 Structure-Fixing Algorithm

4.2.1 Model Formulation

In this section, we offer a different perspective of computing suboptimal traffic tolls. First, recall the formulation (SA-H-R1), the main hardness comes from the bi-linear constraint,

$$f_e^{k(i)} \geq 0,\ u_e^{k(i)} \geq 0,\ f_e^{k(i)} u_e^{k(i)} = 0,\ \forall e, k, i,$$

which is equivalent to

$$f_e^{k(i)} \geq 0,\ u_e^{k(i)} = 0 \ \text{ or } \ f_e^{k(i)} = 0,\ u_e^{k(i)} \geq 0,\ \forall e, k, i.$$

Note that these two conditions are not mutually exclusive since $f_e^{k(i)} = 0,\ u_e^{k(i)} = 0$ lives in both.

When the Gurobi Optimizer solves (SA-H-R1) as a mixed-integer nonlinear programming problem, the solver explores at most $2^{N_E N_M N_S}$ branches, where N_K, N_M, N_S are the number of commodities, edges, and scenarios, respectively. If we know which condition to respect *a prior*, i.e., $f_e^{k(i)} = 0$ or $u_e^{k(i)} = 0$, the bi-linear constraints will degenerate to simple linear constraints. Therefore, a possible way to relax the bi-linear constraint is pre-fixing the condition. To achieve a reasonable relaxation, we first force the following assumption.

Assumption 15 (Flow Structure Fixing Assumption)**.** *The unused links in all scenarios are the same in the pre-toll equilibrium, and remain unused in the post-toll equilibrium.*

Whereas the assumption is somewhat strong from a theoretical standpoint, it makes good sense in practice because the decision maker would not expect toll imposition to significantly alter the link usage. Instead, they are more interested in shifting a portion of people from some routes to others [12].

We are motivated to first compute the pre-toll equilibrium, and record the flows at equilibrium, ${}_{eq}f_e^k$, with superscript (i) discarded because the assumption makes sure the sign stays consistent among all scenarios. The relaxed optimization is then formulated as

$$\min_{H, \boldsymbol{\tau}, \boldsymbol{f}^{(i)}, \boldsymbol{\mu}^{(i)}, \boldsymbol{\lambda}^{(i)}} \ H \tag{SA-H-SF}$$

subject to:

$$\begin{aligned}
&H \geq \sum_e f_e^{(i)} l_e(f_e^{(i)}), \forall i \in \{1, \ldots, N\} \\
&\nabla C(\boldsymbol{f}^{(i)}, \boldsymbol{\tau}) - \boldsymbol{\mu}^{(i)} + \boldsymbol{A}^\top \boldsymbol{\lambda}^{(i)} = \boldsymbol{0} \\
&\boldsymbol{A}\boldsymbol{f}^{(i)} = \boldsymbol{b}^{(i)} \\
&f_e^{k(i)} \geq 0,\ u_e^{k(i)} = 0 \text{ if } {}_{eq}f_e^k \geq 0, \forall e, k \\
&f_e^{k(i)} = 0,\ u_e^{k(i)} \geq 0 \text{ if } {}_{eq}f_e^k = 0, \forall e, k.
\end{aligned}$$

By fixing the traffic flow structure, we turn the non-convex scenario optimization into a convex one (SA-H-SF), which can be solved very efficiently. More interestingly, the decision maker will be able to comprise between robustness and performance by applying techniques provided by [10].

4.2.2 Preliminary Result Analysis

To better illustrate the performance of the structure-fixing algorithm, we consider the deterministic case with the nominal demands and solve the optimization (NLP) with the structure-fixing adaption.

$$\min_{H,\boldsymbol{\tau},\boldsymbol{f},\boldsymbol{\mu},\boldsymbol{\lambda}} \sum_e f_e l_e(f_e) \tag{NLP-SF}$$

subject to:

$$\begin{aligned}
&\nabla C(\boldsymbol{f},\boldsymbol{\tau}) - \boldsymbol{\mu} + \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{0} \\
&\boldsymbol{A}\boldsymbol{f} = \boldsymbol{b} \\
&f_e^k \geq 0,\ u_e^k = 0 \text{ if } {}_{eq}f_e^k \geq 0, \forall e, k \\
&f_e^k = 0,\ u_e^k \geq 0 \text{ if } {}_{eq}f_e^k = 0, \forall e, k.
\end{aligned}$$

Take the Braess's network for example, we scale the nominal demand by a scaling factor, and compare the price of anarchy with and without tolls, obtained with the structure-fixing algorithm.

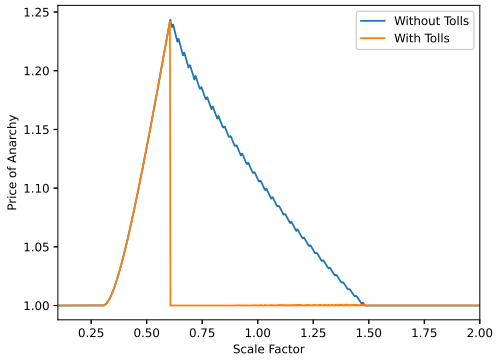


Figure 4.19: Structure-Fixing Tolls on Braess's Network

Figure 4.19 shows structure-fixing tolls can incentivize the optimal traffic flows for any scale factor greater than 0.606 because Assumption 15 does not exclude the optimal solution in this case. However, for a scale factor smaller than 0.606, the algorithm was not able to improve the traffic efficiency, since edge 2 and 3 in Figure 2.3 were not used in the pre-toll equilibrium and thus we could not use them later.

We can also obtain the break point 0.606 mathematically. Given a small scale factor α, the users would like to choose the path via edges $1, 4, 5$. When some players start to use the path via edges $1, 3$, the costs along these two paths should be the same, per Definition 5:

$$\begin{aligned}
l_1(f_1) + l_4(f_4) + l_5(f_5) &= l_1(f_1) + l_3(f_3) \\
l_1(\alpha d) + l_4(\alpha d) + l_5(\alpha d) &= l_1(\alpha d) + l_3(0) \\
10\alpha d + 10\alpha d + 10 + 10\alpha d &= 10\alpha d + 50 \\
\alpha &= \frac{40}{11d} = 0.606
\end{aligned}$$

Note that the fluctuation of the curve in Figure 4.19 comes from the numerical error.

Next, we consider a more complex network - the Sioux Falls network. We scale all the demands and illustrate in Figure 4.20 the price of anarchy with and without the structuring-fixing tolls.

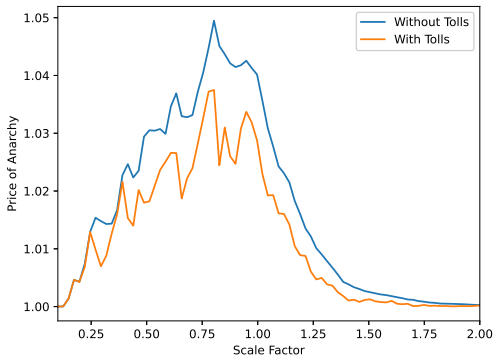


Figure 4.20: Structure-Fixing Tolls on Sioux Falls Network

We can observe that the performance is quite limited but still the "With Tolls" curve is never below the "Without Tolls" one. In other words, in the deterministic case, structure-fixing tolls can at least remain PoA, if cannot improve. The fact can be simply explained - "Without Tolls" is a feasible point of (NLP-SF) and certainly results in a suboptimal objective, while "With Tolls" gives the optimal objective of (NLP-SF).

4.2.3 Discussion

The unsatisfactory performance of (NLP-SF) comes from the strong Assumption 15, which prohibits us from exploring numerous possible candidates, i.e., performance is sacrificed for efficiency. Due to the poor performance, it is of little value to continue to the scenario optimization. Since (SA-H-SF) is almost equivalently to one node in (SA-H-R1), future work is to design a sophisticated scheme to locate potentially desirable nodes, i.e., specially designed branching scheme. The evolutionary algorithm may be a good starting point to design a branch-exploring strategy.

4.3 Discussion

Besides the two algorithms proposed in this chapter, we also tested the existing numerical optimization algorithms to compute a locally optimal solution. For example, trust-region constrained algorithm and sequential least squares programming (SLSQP) algorithm.

Trust-region constrained algorithm requires the gradient and Hessian matrix of the objective function, as well as the Jacobian and Hessian matrices of the constraints. For the scenario optimization, we can easily obtain these required components and start the algorithm. However, the computational time is unacceptably long even for a small network. Moreover, the locally optimal solution performs even worse than no tolls at all. The reason is that the scenario optimization is too complex with many decision variables and constraints. The decision variables have complicated impacts on the objective function, and thus the locally optimal solution behaves in an undesirable way.

SLSQP requires only the Jacobian matrix of the objective function and constraints. As a sequential algorithm, it outperforms trust-region constrained algorithm in computational efficiency. However, SLSQP does not deal with the cases involving singular matrices. As for the performance, SLSQP provides a solution as poor as the trust-region constrained algorithm.

Chapter 5

Conclusion and Outlook

Congestion pricing has been a popular topic for policy makers. In this thesis, we propose several algorithms to compute the probabilistically robust tolls based on the collected scenarios. We also illustrate the performance and robustness of these tolls under uncertainties. We highlight that the numerical gradient descent algorithm will be of most value in practice due to its satisfying efficiency and performance. Specifically, on the Sioux Falls network, where demands vary within 5%, we can draw the conclusion "with the computed tolls, the probability that the PoA exceeds 1.02 is no larger than 0.295 with confidence $1 - 10^{-6}$" given 100 previously-observed scenarios.

Besides the algorithms mentioned in this thesis, there may exist other algorithms to compute the robust optimal tolls. We would like to offer some possible directions for interested researchers:

- One main reason for the computational hardness is that the social cost cannot be expressed as an analytical function of the tolls. Deep learning techniques, as one of the most popular tools these days, may be utilized to approximate the hidden function with neural networks. If the approximation is accurate enough, the numerical gradient descent algorithm can be run much faster and more accurately. Similarly, multi-variate regression is also an option.
- The toll design task falls into the category of data-driven decision making problems. In practice, the feasible region of tolls is usually discrete. With the decision space well-defined, unsupervised reinforcement learning is potentially another technique to deal with the toll design task.
- In this thesis, we study the performance and robustness of *constant-value* tolls, i.e, the charged tolls are just constant and independent of the on-link flows. The future work is to investigate how to obtain probabilistically robust *linear* tolls and *polynomial* tolls.
- In the case that we do not want to tax all roads, which ones should be taxed? In practice, the policy makers tend to tax the main roads and highways. However, theoretically it is impossible to efficiently detect the edges most responsible for the inefficiency [36]. We thus expect future research to investigate how to decide links to be taxed.
- The computational hardness of computing robust tolls comes from the natural of bi-level optimization. (SA-H) transfers the lower level optimization with KKT conditions, but does not save us from the NP-hardness. However, in the case of a parallel-arc network with affine latency functions, we have the potential to exploit the network structure to improve the algorithm efficiency - [25, 3] propose algorithms to formulate the toll design task as a couple of convex optimizations, which can be easily solved. It is possible to combine their algorithms and the scenario approach to divide the NP-hard problem into several simple ones and conquer them one by one.

Bibliography

[1] G. B. Allende and G. Still. Solving bilevel programs with the kkt-approach. *Mathematical programming*, 138(1):309–332, 2013.

[2] M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation*. Rand Corporation, 1956.

[3] V. Bonifaci, M. Salek, and G. Schäfer. Efficiency of restricted tolls in non-atomic network routing games. In *International Symposium on Algorithmic Game Theory*, pages 302–313. Springer, 2011.

[4] P. N. Brown and J. R. Marden. Optimal mechanisms for robust coordination in congestion games. *IEEE Transactions on Automatic Control*, 63(8):2437–2448, 2017.

[5] P. N. Brown and J. R. Marden. Studies on robust social influence mechanisms: Incentives for efficient network routing in uncertain settings. *IEEE Control Systems Magazine*, 37(1):98–115, 2017.

[6] G. Calafiore and M. C. Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming*, 102(1):25–46, 2005.

[7] G. C. Calafiore and M. C. Campi. The scenario approach to robust control design. *IEEE Transactions on automatic control*, 51(5):742–753, 2006.

[8] M. C. Campi and S. Garatti. The exact feasibility of randomized solutions of uncertain convex programs. *SIAM Journal on Optimization*, 19(3):1211–1230, 2008.

[9] M. C. Campi and S. Garatti. Wait-and-judge scenario optimization. *Mathematical Programming*, 167(1):155–189, 2018.

[10] M. C. Campi and S. Garatti. Scenario optimization with relaxation: a new tool for design and application to machine learning problems. *arXiv preprint arXiv:2004.05839*, 2020.

[11] M. C. Campi, S. Garatti, and F. A. Ramponi. A general scenario theory for nonconvex optimization and decision making. *IEEE Transactions on Automatic Control*, 63(12):4067–4078, 2018.

[12] M. Chen, D. H. Bernstein, S. I. Chien, and K. C. Mouskos. Simplified formulation of the toll design problem. *Transportation Research Record*, 1667(1):88–95, 1999.

[13] R. Colini-Baldeschi, M. Klimm, and M. Scarsini. Demand-independent optimal tolls. *arXiv preprint arXiv:1708.02737*, 2017.

[14] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

[15] J. R. Correa, A. S. Schulz, and N. E. Stier-Moses. On the inefficiency of equilibria in congestion games. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 167–181. Springer, 2005.

[16] P. Dubey and J. D. Rogawski. Inefficiency of nash equilibria in strategic market games. *BEBR faculty working paper; no. 1104*, 1985.

[17] P. Ferrari. Road network toll pricing and social welfare. *Transportation Research Part B: Methodological*, 36(5):471–483, 2002.

[18] L. Fleischer. Linear tolls suffice: New bounds and algorithms for tolls in single source networks. *Theoretical Computer Science*, 348(2-3):217–225, 2005.

[19] M. Frank, P. Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[20] S. Garatti and M. Campi. Risk and complexity in scenario optimization. *Mathematical Programming*, pages 1–37, 2019.

[21] L. M. Gardner, A. Unnikrishnan, and S. T. Waller. Robust pricing of transportation networks under uncertain demand. *Transportation Research Record*, 2085(1):21–30, 2008.

[22] L. M. Gardner, A. Unnikrishnan, and S. T. Waller. Solution methods for robust pricing of transportation networks under uncertain demand. *Transportation Research Part C: Emerging Technologies*, 18(5):656–667, 2010.

[23] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2021.

[24] T. Harks, I. Kleinert, M. Klimm, and R. H. Möhring. Computing network tolls with support constraints. *Networks*, 65(3):262–285, 2015.

[25] M. Hoefer, L. Olbrich, and A. Skopalik. Taxing subnetworks. In *International Workshop on Internet and Network Economics*, pages 286–294. Springer, 2008.

[26] T. Jelinek, M. Klaas, and G. Schäfer. Computing optimal tolls with arc restrictions and heterogeneous players. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[27] R. G. Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical programming*, 32(2):146–164, 1985.

[28] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 404–413. Springer, 1999.

[29] M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management science*, 44(12-part-1):1608–1622, 1998.

[30] M. Labbé and A. Violin. Bilevel programming and price setting problems. *4OR*, 11(1):1–30, 2013.

[31] U. B. of Public Roads. Office of Planning. Urban Planning Division. *Traffic Assignment Manual for Application with a Large, High Speed Computer*. US Department of Commerce, 1964.

[32] C. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753, 2001.

[33] A. C. Pigou. *The Economics of Welfare*. London: Macmillan, 1920.

[34] A. Rapoport. Prisoner's dilemma. In *Game Theory*, pages 199–204. Springer, 1989.

[35] T. Reed. Inrix global traffic scorecard. *INRIX research*, 2019.

[36] T. Roughgarden. Designing networks for selfish users is hard. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 472–481. IEEE, 2001.

[37] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.

[38] T. Roughgarden and E. Tardos. Introduction to the inefficiency of equilibria. *Algorithmic game theory*, 17:443–459, 2007.

[39] D. Schrank, B. Eisele, and T. Lomax. Tti's 2012 urban mobility report. *Texas A&M Transportation Institute. The Texas A&M University System*, 4, 2012.

[40] A. Sinha, P. Malo, and K. Deb. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017.

[41] Transportation Networks for Research Core Team. Transportation networks for research. `https://github.com/bstabler/TransportationNetworks`, 2021.

[42] R. Turvey. Marginal cost. *The Economic Journal*, 79(314):282–299, 1969.

[43] E. T. Verhoef. Second-best congestion pricing in general networks. heuristic algorithms for finding second-best optimal toll levels and toll points. *Transportation Research Part B: Methodological*, 36(8):707–729, 2002.

[44] L. N. Vicente and P. H. Calamai. Bilevel and multilevel programming: A bibliography review. *Journal of Global optimization*, 5(3):291–306, 1994.

[45] C. Wang, Q. Tian, and H.-J. Huang. Inefficiency of marginal-cost tolls in transportation networks with stochastic demands. In *2017 International Conference on Service Systems and Service Management*, pages 1–4. IEEE, 2017.

[46] J. G. Wardrop. Road paper. some theoretical aspects of road traffic research. *Proceedings of the institution of civil engineers*, 1(3):325–362, 1952.

[47] H. Yan and W. H. Lam. Optimal road tolls under conditions of queueing and congestion. *Transportation Research Part A: Policy and Practice*, 30(5):319–332, 1996.

[48] H. Yang and S. Yagar. Traffic assignment and traffic control in general freeway-arterial corridor systems. *Transportation Research Part B: Methodological*, 28(6):463–486, 1994.

[49] H. Yang, S. Yagar, Y. Iida, and Y. Asakura. An algorithm for the inflow control problem on urban freeway networks with user-optimal flows. *Transportation Research Part B: Methodological*, 28(2):123–139, 1994.

[50] H. Yang and X. Zhang. Optimal toll design in second-best link-based congestion pricing. *Transportation Research Record*, 1857(1):85–92, 2003.