Differentiable Equilibrium Computation with Decision Diagrams for Stackelberg Models of Combinatorial Congestion Games

Shinsaku Sakaue
The University of Tokyo
Tokyo, Japan
sakaue@mist.i.u-tokyo.ac.jp

Kengo Nakamura
NTT Communication Science Laboratories
Kyoto, Japan
kengo.nakamura.dx@hco.ntt.co.jp

Abstract

We address Stackelberg models of combinatorial congestion games (CCGs); we aim to optimize the parameters of CCGs so that the selfish behavior of non-atomic players attains desirable equilibria. This model is essential for designing such social infrastructures as traffic and communication networks. Nevertheless, computational approaches to the model have not been thoroughly studied due to two difficulties: (I) bilevel-programming structures and (II) the combinatorial nature of CCGs. We tackle them by carefully combining (I) the idea of *differentiable* optimization and (II) data structures called *zero-suppressed binary decision diagrams* (ZDDs), which can compactly represent sets of combinatorial strategies. Our algorithm numerically approximates the equilibria of CCGs, which we can differentiate with respect to parameters of CCGs by automatic differentiation. With the resulting derivatives, we can apply gradient-based methods to Stackelberg models of CCGs. Our method is tailored to induce Nesterov's acceleration and can fully utilize the empirical compactness of ZDDs. These technical advantages enable us to deal with CCGs with a vast number of combinatorial strategies. Experiments on real-world network design instances demonstrate the practicality of our method.

1 Introduction

Congestion games (CGs) [49] form an important class of non-cooperative games and appear in various resource allocation scenarios. Combinatorial CGs (CCGs) can model more complex situations where each strategy is a combination of resources. A well-known example of a CCG is selfish routing [50], where each player on a traffic network chooses an origin-destination path, which is a strategy given by a combination of some roads with limited width (resources). Computing the outcomes of players' selfish behaviors (or equilibria) is essential when designing social infrastructures such as traffic networks. Therefore, how to compute equilibria of CCGs has been widely studied [5, 12, 57, 42].

In this paper, we are interested in the perspective of the leader who designs non-atomic CCGs. For example, the leader aims to optimize some traffic-network parameters (e.g., road width values) so that players can spend less traveling time at equilibrium. An equilibrium of non-atomic CCGs is characterized by an optimum of potential function minimization [41, 53]. Thus, designing CCGs can be seen as a Stackelberg game; the leader optimizes the parameters of CCGs to minimize an objective function (typically, the social-cost function) while the follower, who represents the population of selfish non-atomic players, minimizes the potential function. This mathematical formulation is called the Stackelberg model in the context of traffic management [46]. Therefore, we call our model with general combinatorial strategies a Stackelberg model of CCGs.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Stackelberg models of CCGs have been studied for cases where the potential function minimization has desirable properties. For example, Patriksson and Rockafellar [46] proposed a descent algorithm for traffic management using the fact that projections onto flow polyhedra can be done efficiently. However, many practical CCGs have more complicated structures. For example, in communication network design, each strategy is given by a Steiner tree (see [24, 42] and Section 1.1), and thus the projection (and even optimizing linear functions) is NP-hard. How to address such computationally challenging Stackelberg models of CCGs has not been well studied, despite its practical importance.

Inspired by a recent equilibrium computation method [42], we tackle the combinatorial nature of CCGs by representing their strategy sets with *zero-suppressed binary decision diagrams* (ZDDs) [39, 32], which are well-established data structures that provide empirically compact representations of combinatorial objects (e.g., Steiner trees and Hamiltonian paths). Although the previous method [42] can efficiently approximate equilibria with a Frank–Wolfe-style algorithm [18], its computation procedures break the differentiability of the outputs in the CCG parameters (the leader's variables), preventing us from obtaining gradient information required for optimizing leader's objective functions.

Our contribution is to develop a *differentiable* pipeline from leader's variables to equilibria of CCGs, thus enabling application of gradient-based methods to the Stackelberg models of CCGs. We smooth the Frank–Wolfe iterations using softmin, thereby making computed equilibria differentiable with respect to the leader's variables by automatic differentiation (or backpropagation). Although the idea of smoothing with softmin is prevalent [29, 38], our method has the following technical novelty:

- Our algorithm is tailored to induce Nesterov's acceleration, making both equilibrium computation and backpropagation more efficient. To the best of our knowledge, the idea of simultaneously making iterative optimization methods both differentiable and faster is new.
- Our method consists of simple arithmetic operations performed with ZDDs as in Algorithm 2. This is essential for making our equilibrium computation accept automatic differentiation. The per-iteration complexity of our method is linear in the ZDD size.

Armed with these advantages, our method can work with CCGs that have an enormous number of combinatorial strategies. We experimentally demonstrate its practical usefulness in real-world network design instances. Our method brings benefits by improving the designs of social infrastructures.

Notation. Let $[n] \coloneqq \{1, \dots, n\}$. For any $S \subseteq [n]$, $\mathbf{1}_S \in \{0,1\}^n$ denotes a binary vector whose i-th entry is 1 if and only if $i \in S$. Let $\|\cdot\|$ be the ℓ_2-norm.

1.1 Problem setting

We introduce the problem setting and some assumptions. For simplicity, we describe the *symmetric* setting, although our method can be extended to an *asymmetric* setting, as explained in Appendix A.

Combinatorial congestion games (CCGs). Suppose that there is an infinite amount of players with an infinitesimal mass (i.e., non-atomic). We assume the total mass is 1 without loss of generality. Let $[n]$ be a set of resources and let $\mathcal{S} \subseteq 2^{[n]}$ be a set of all feasible strategies. We define $d \coloneqq |\mathcal{S}|$, which is generally exponential in n. Each player selects strategy $S \in \mathcal{S}$. Let $\boldsymbol{y} \in [0,1]^n$ be a vector whose i-th entry indicates the total mass of players using resource $i \in [n]$. In other words, if we let $\boldsymbol{z} \in \triangle^d$ be a vector whose entry z_S ($S \in \mathcal{S}$) indicates the total mass of players choosing S, we have $\boldsymbol{y} \coloneqq \sum_{S \in \mathcal{S}} z_S \mathbf{1}_S \in \mathbb{R}^n$. Therefore, $\boldsymbol{y}$ is included in convex hull $\mathcal{C} \coloneqq \left\{ \sum_{S \in \mathcal{S}} z_S \mathbf{1}_S \;\middle|\; \boldsymbol{z} \in \triangle^d \right\}$, where $\triangle^d \coloneqq \left\{ \boldsymbol{z} \in \mathbb{R}^d \;\middle|\; \boldsymbol{z} \geq 0, \sum_{S \in \mathcal{S}} z_S = 1 \right\}$ is the $(d-1)$-dimensional probability simplex. A player choosing S incurs cost $c_S(\boldsymbol{y}) \coloneqq \sum_{i \in S} c_i(y_i)$, where each $c_i : \mathbb{R} \to \mathbb{R}$ is assumed to be strictly increasing; this corresponds to a natural situation where cost c_i increases as $i \in [n]$ becomes more congested. Each player selfishly selects a strategy to minimize his/her own cost.

Equilibrium and potential functions. We say $\boldsymbol{z} \in \triangle^d$ attains a *(Wardrop) equilibrium* if for every $S \in \mathcal{S}$ such that $z_S > 0$, it holds that $c_S(\boldsymbol{y}) \leq \min_{S' \in \mathcal{S}} c_{S'}(\boldsymbol{y})$, where $\boldsymbol{y} \coloneqq \sum_{S \in \mathcal{S}} z_S \mathbf{1}_S$. That is, no one has an incentive to deviate unilaterally. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a *potential function* defined as $f(\boldsymbol{y}) \coloneqq \sum_{i \in [n]} \int_0^{y_i} c_i(u) \mathrm{d}u$. From the first-order optimality condition, it holds that $\boldsymbol{z}$ attains an equilibrium iff $\boldsymbol{y} = \sum_{S \in \mathcal{S}} z_S \mathbf{1}_S$ satisfies $\boldsymbol{y} = \operatorname{argmin}_{\boldsymbol{u} \in \mathcal{C}} f(\boldsymbol{u})$. Note that minimizer $\boldsymbol{y}$ is unique since c_i is strictly increasing, which means f is strictly convex (not necessarily strongly convex).

Stackelberg model of CCGs. We turn to the problem of designing CCGs. For $i \in [n]$, let $c_i(y_i; \boldsymbol{\theta})$ be a cost function with parameters $\boldsymbol{\theta} \in \Theta$. We assume $c_i(y_i; \boldsymbol{\theta})$ to be strictly increasing in y_i for any $\boldsymbol{\theta} \in \Theta$ and differentiable with respect to $\boldsymbol{\theta}$ for any $\boldsymbol{y} \in \mathcal{C}$. Let $f(\boldsymbol{y}; \boldsymbol{\theta}) = \sum_{i \in [n]} \int_0^{y_i} c_i(u; \boldsymbol{\theta}) \mathrm{d}u$ be a parameterized potential function, which is strictly convex in $\boldsymbol{y}$ for any $\boldsymbol{\theta} \in \Theta$. A leader who designs CCGs aims to optimize $\boldsymbol{\theta}$ values so that an objective function, $F : \Theta \times \mathcal{C} \to \mathbb{R}$, is minimized at an equilibrium of CCGs. Typically, F is a social-cost function defined as $F(\boldsymbol{\theta}, \boldsymbol{y}) = \sum_{i \in [n]} c_i(y_i; \boldsymbol{\theta}) y_i$, which represents the total cost incurred by all players. Since an equilibrium is characterized as a minimizer of potential function f, the leader's problem can be written as follows:

$$\underset{\boldsymbol{\theta} \in \Theta}{\text{minimize}} \quad F(\boldsymbol{\theta}, \boldsymbol{y}) \qquad \text{subject to} \quad \boldsymbol{y} = \underset{\boldsymbol{u} \in \mathcal{C}}{\operatorname{argmin}} f(\boldsymbol{u}; \boldsymbol{\theta}). \tag{1}$$

Since minimizer $\boldsymbol{y}(\boldsymbol{\theta}) \coloneqq \boldsymbol{y}$ is unique, we can regard $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ as a function of $\boldsymbol{\theta}$. We study how to approximate derivatives of $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ with respect to $\boldsymbol{\theta}$ for applying gradient-based methods to (1).

Example 1: traffic management. We are given a network with an origin-destination (OD) pair. Let $[n]$ be the edge set and let $\mathcal{S} \subseteq 2^{[n]}$ be the set of all OD paths. Each edge in the network has cost function $c_i(y_i; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ controls the width of the roads (edges). A natural example of the cost functions is $c(y_i; \boldsymbol{\theta}) = y_i / \theta_i$ for $\theta_i > 0$ (see, e.g., [46]), which satisfies the above assumptions, i.e., strictly increasing in y_i and differentiable in θ_i. Once $\boldsymbol{\theta}$ is fixed, players selfishly choose OD paths and consequently reach an equilibrium. The leader wants to find $\boldsymbol{\theta}$ that minimizes social cost F at equilibrium, which can be formulated as a Stackelberg model of form (1). Note that although the Stackelberg model of standard selfish routing is well studied [46], there are various variants (e.g., routing with budget constraints [28, 42]) for which existing methods do not work efficiently.

Example 2: communication network design. We consider a situation where multi-site meetings are held on a communication network (see, e.g., [24, 42]). Given an undirected network with edge set $[n]$ and some vertices called terminals, groups of people at terminals hold multi-site meetings, including people at all the terminals. Since each group wants to minimize the communication delays caused by congestion, each selfishly chooses a way to connect all the terminals, namely, a Steiner tree covering all the terminals. If we let $c_i(y_i; \boldsymbol{\theta})$ indicate the delay of the i-th edge, a group choosing Steiner tree $S \in \mathcal{S}$ incurs cost $c_S(\boldsymbol{y}; \boldsymbol{\theta})$. As with the above traffic-management example, the problem of optimizing $\boldsymbol{\theta}$ to minimize the total delay at equilibrium can be written as (1).

1.2 Related work

Problems of form (1) arise in many fields, e.g., Stackelberg games [55], mathematical programming with equilibrium constraints [36], and bilevel programming [10, 13], which have been gaining attention in machine learning [17, 15]. Optimization problems with bilevel structures are NP-hard in most cases [22] (tractable cases include, e.g., when follower's problems are unconstrained and strongly convex [19], which does not hold in our case). Thus, how to apply gradient-based methods occupies central interest [15, 20]. In our Stackelberg model of CCGs, in addition to the bilevel structure, the follower's problem is defined on combinatorial strategy sets $\mathcal{S}$, further complicating it. Therefore, unlike the above studies, we focus on how to address such difficult problems by leveraging computational tools, including ZDDs [39] and automatic differentiation [34, 21].

Stackelberg models often arise in traffic management. Although many existing studies [46, 35, 6, 9] analyze theoretical aspects utilizing instance-specific structures (e.g., compact representations of flow polyhedra), applications to other types of realistic CCGs remain unexplored. By contrast, as with the previous method [42], our method is built on versatile ZDD representations of strategy sets, and the derivatives with respect to CCG parameters can be automatically computed with backpropagation. Thus, compared to the methods studied in traffic management, ours can be easily applied to and works efficiently with a broad class of realistic CCGs with complicated combinatorial strategies.

Our method is inspired by the emerging line of work on differentiable optimization [4, 60, 2, 51]. For differentiating outputs with respect to the parameters of optimization problems, two major approaches have been studied [20]: *implicit* and *iterative* differentiation. The first approach applies the implicit function theorem to equation systems derived from the Karush–Kuhn–Tucker (KKT) condition (akin to the single-level reformulation approach to bilevel programming). In our case, this approach is too expensive since the combinatorial nature of CCGs generally makes the KKT equation system exponentially large [16]. Our method is categorized into the second approach, which computes

a numerical approximation of an optimum with iterations of differentiable steps. This idea has yielded success in many fields [14, 37, 45, 7, 17]. Concerning combinatorial optimization, although differentiable methods for linear objectives are well studied [38, 60, 47, 8], no differentiable method has been developed for convex minimization on polytopes of, e.g., Steiner trees or Hamiltonian paths; this is what we need for dealing with the potential function minimization of CCGs. To this end, we use a Frank–Wolfe-style algorithm and ZDD representations of combinatorial objects.

2 Differentiable iterative equilibrium computation

We consider applying gradient-based methods (e.g., projected gradient descent) to problem (1). To this end, we need to compute the following gradient with respect to $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^k$ in each iteration:

$$\nabla F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta})) = \nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta})) + \nabla \boldsymbol{y}(\boldsymbol{\theta})^\top \nabla_{\boldsymbol{y}} F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta})),$$

where $\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ and $\nabla_{\boldsymbol{y}} F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ denote the gradients with respect to the first and second arguments, respectively, and $\nabla \boldsymbol{y}(\boldsymbol{\theta})$ is the $n \times k$ Jacobian matrix.[1] The computation of $\nabla \boldsymbol{y}(\boldsymbol{\theta})$ is the most challenging part and requires differentiating $\boldsymbol{y}(\boldsymbol{\theta}) = \operatorname{argmin}_{\boldsymbol{u} \in \mathcal{C}} f(\boldsymbol{u}; \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. We employ the iterative differentiation approach for efficiently approximating $\nabla \boldsymbol{y}(\boldsymbol{\theta})$.

2.1 Technical overview

For computing equilibrium $\boldsymbol{y}(\boldsymbol{\theta})$, Nakamura et al. [42] solved potential function minimization with a variant of the Frank–Wolfe algorithm [33], whose iterations can be performed efficiently by using compact ZDD representations of combinatorial strategies. To the best of our knowledge, no other equilibrium computation methods can deal with various CCGs that have complicated combinatorial strategies, e.g., Steiner trees. Hence we build on [42] and extend their method to Stackelberg models.

First, we review the standard Frank–Wolfe algorithm. Starting from $\boldsymbol{x}_0 \in \mathcal{C}$, it alternately computes $\boldsymbol{s}_t = \operatorname{argmin}_{\boldsymbol{s} \in \mathcal{C}} \langle \nabla f(\boldsymbol{x}_t; \boldsymbol{\theta}), \boldsymbol{s} \rangle$ and $\boldsymbol{x}_{t+1} = (1 - \gamma_t)\boldsymbol{x}_t + \gamma_t \boldsymbol{s}_t$, where γ_t is conventionally set to $\frac{2}{t+2}$. As shown in [18, 27], $\boldsymbol{x}_T$ has an objective error of $\mathrm{O}(1/T)$. Thus, we can obtain numerical approximation $\boldsymbol{y}_T(\boldsymbol{\theta}) = \boldsymbol{x}_T$ of equilibrium $\boldsymbol{y}(\boldsymbol{\theta})$ such that $f(\boldsymbol{y}_T(\boldsymbol{\theta}); \boldsymbol{\theta}) - f(\boldsymbol{y}(\boldsymbol{\theta}); \boldsymbol{\theta}) \leq \mathrm{O}(1/T)$. For obtaining gradient $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$, however, the above Frank–Wolfe algorithm does not work (neither does its faster variant used in [42]). This is because $\boldsymbol{s}_t = \operatorname{argmin}_{\boldsymbol{s} \in \mathcal{C}} \langle \nabla f(\boldsymbol{x}_t; \boldsymbol{\theta}), \boldsymbol{s} \rangle$ is piecewise constant in $\boldsymbol{\theta}$, which makes $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$ zero almost everywhere and undefined at some $\boldsymbol{\theta}$.

To resolve this issue, we develop a differentiable Frank–Wolfe algorithm by using softmin. We denote the softmin operation by $\boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c})$ (detailed below). Since softmin can be seen as a differentiable proxy for argmin, one may simply replace $\boldsymbol{s}_t = \operatorname{argmin}_{\boldsymbol{s} \in \mathcal{C}} \langle \nabla f(\boldsymbol{x}_t; \boldsymbol{\theta}), \boldsymbol{s} \rangle$ with $\boldsymbol{s}_t = \boldsymbol{\mu}_{\mathcal{S}}(\eta_t \nabla f(\boldsymbol{x}_t; \boldsymbol{\theta}))$, where $\eta_t > 0$ is a scaling factor. Actually, the modified algorithm yields an $\mathrm{O}(1/T)$ convergence by setting $\eta_t = \Omega(t)$ (see [27, Theorem 1]). This modification, however, often degrades the empirical convergence of the Frank–Wolfe algorithm, as demonstrated in Section 4.1. We, therefore, consider leveraging softmin for acceleration while keeping the iterations differentiable. Based on an accelerated Frank–Wolfe algorithm [59], we compute $\boldsymbol{y}_T(\boldsymbol{\theta})$ as in Algorithm 1, and obtain $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$ by applying backpropagation. Furthermore, in Section 3, we explain how to efficiently compute $\boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c})$ by using a ZDD-based technique [52]; importantly, its computation procedure also accepts the backpropagation.

While our work is built on the existing methods [59, 52, 42], none of them are intended to develop differentiable methods. A conceptual novelty of our work is its careful combination of those methods for developing a differentiable and accelerated optimization method, with which we can compute $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$. This enables the application of gradient-based methods to the Stackelberg models of CCGs.

2.2 Details of Algorithm 1

We compute $\boldsymbol{y}_T(\boldsymbol{\theta})$ with Algorithm 1. Note that $\boldsymbol{s}_t$, $\boldsymbol{c}_t$, and $\boldsymbol{x}_t$ depend on $\boldsymbol{\theta}$, which is not explicitly indicated for simplicity. The most crucial part is Step 5, where we use softmin rather than argmin to

[1] Although the derivatives of $\boldsymbol{y}(\boldsymbol{\theta})$ may not be unique, we abuse the notation and write $\nabla \boldsymbol{y}(\boldsymbol{\theta})$ for simplicity. As we will see shortly, we numerically approximate $\boldsymbol{y}(\boldsymbol{\theta})$ with $\boldsymbol{y}_T(\boldsymbol{\theta})$ whose derivative, $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$, exists uniquely. Therefore, when discussing our iterative differentiation method, we can ignore the abuse of notation.

Algorithm 1 Differentiable Frank–Wolfe-based equilibrium computation

$\boldsymbol{s}_0 = \boldsymbol{c}_0 = \boldsymbol{0}$, $\boldsymbol{x}_{-1} = \boldsymbol{x}_0 = \boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c}_0)$, and $\alpha_t = t$ $(t = 0, \dots, T)$
for $t = 1, \dots, T$ **:**
  $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} - \alpha_{t-1}\boldsymbol{x}_{t-2} + (\alpha_{t-1} + \alpha_t)\boldsymbol{x}_{t-1}$
  $\boldsymbol{c}_t = \boldsymbol{c}_{t-1} + \eta\alpha_t \nabla f\left(\frac{2}{t(t+1)}\boldsymbol{s}_t; \boldsymbol{\theta}\right)$ ▷ $\nabla f(\boldsymbol{y}; \boldsymbol{\theta})_i = c_i(y_i; \boldsymbol{\theta})$ is differentiable in $\boldsymbol{\theta}$
  Compute $\boldsymbol{x}_t = \boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c}_t)$ with Algorithm 2 ▷ Differentiable softmin computation
return $\boldsymbol{y}_T(\boldsymbol{\theta}) = \frac{2}{T(T+1)} \sum_{t=1}^{T} \alpha_t \boldsymbol{x}_t$

make the output differentiable in $\boldsymbol{\theta}$. Specifically, given any $\boldsymbol{c} \in \mathbb{R}^n$, we compute $\boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c})$ as follows:

$$\boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c}) \coloneqq \sum_{S \in \mathcal{S}} \mathbf{1}_S \frac{\exp\left(-\boldsymbol{c}^\top \mathbf{1}_S\right)}{\sum_{S' \in \mathcal{S}} \exp\left(-\boldsymbol{c}^\top \mathbf{1}_{S'}\right)}.$$

Intuitively, each entry in $\boldsymbol{c}$ represents the cost of each $i \in [n]$, and $\boldsymbol{c}^\top \mathbf{1}_S$ represents the cost of $S \in \mathcal{S}$. We consider a probability distribution over $\mathcal{S}$ defined by softmin with respect to costs $\{\boldsymbol{c}^\top \mathbf{1}_S\}_{S \in \mathcal{S}}$, and then marginalize it. The resulting vector is a convex combination of $\{\mathbf{1}_S\}_{S \in \mathcal{S}}$ and thus always included in $\mathcal{C}$. In the context of graphical modeling, this operation is called marginal inference [58]. Here, $\boldsymbol{\mu}_{\mathcal{S}}$ is defined by a summation over $\mathcal{S}$, and explicitly computing it is prohibitively expensive. Section 3 details how to efficiently compute $\boldsymbol{\mu}_{\mathcal{S}}$ by leveraging the ZDD representations of $\mathcal{S}$.

2.3 Convergence guarantee

Algorithm 1 is designed to induce Nesterov's acceleration [43, 59] and achieves an $\mathrm{O}(1/T^2)$ convergence, which is faster than the $\mathrm{O}(1/T)$ convergence of the original Frank–Wolfe algorithm.

Theorem 1. *Fix $\boldsymbol{\theta} \in \Theta$ and assume $f(\cdot; \boldsymbol{\theta})$ to be L-smooth on $\mathbb{R}^d$, i.e., $\Phi(\boldsymbol{z}) \coloneqq f(\sum_{S \in \mathcal{S}} z_S \mathbf{1}_S; \boldsymbol{\theta})$ $(\forall \boldsymbol{z} \in \mathbb{R}^d)$ satisfies $\Phi(\boldsymbol{z}') \leq \Phi(\boldsymbol{z}) + \langle \nabla\Phi(\boldsymbol{z}), \boldsymbol{z}' - \boldsymbol{z} \rangle + \frac{L}{2}\|\boldsymbol{z}' - \boldsymbol{z}\|^2$ for all $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{R}^d$.[2] If we let $\eta \in [\frac{1}{CL}, \frac{1}{4L}]$ for some $C > 4$, Algorithm 1 returns $\boldsymbol{y}_T(\boldsymbol{\theta})$ such that*

$$f(\boldsymbol{y}_T(\boldsymbol{\theta}); \boldsymbol{\theta}) - f(\boldsymbol{y}(\boldsymbol{\theta}); \boldsymbol{\theta}) \leq \mathrm{O}\left(\frac{CL \ln d}{T^2}\right).$$

We present the proof in Appendix B. In essence, softmin can be seen as a dual mirror descent step with the Kullback–Leibler divergence, and combining it with a primal gradient descent step yields the acceleration [3]. Although the acceleration technique itself is well studied, it has not been explored in the context of differentiable optimization. To the best of our knowledge, simultaneously making iterative optimization methods both differentiable and faster is a novel idea. This observation can be beneficial for developing other fast differentiable iterative algorithms. Experiments in Section 4.1 demonstrate that the acceleration indeed enhances the convergence speed in practice.

Note that the faster convergence enables us to more efficiently compute both $\boldsymbol{y}_T(\boldsymbol{\theta})$ and $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$. The latter is because Algorithm 1 with a smaller T generates a smaller computation graph, which determines the computation complexity of the backpropagation for obtaining $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$. Therefore, Algorithm 1 is suitable as an efficient differentiable pipeline between $\boldsymbol{\theta}$ and $\boldsymbol{y}_T(\boldsymbol{\theta})$.

2.4 Implementation consideration: how to choose η and T

While Theorem 1 suggests setting η to $\frac{1}{4L}$ or less, this choice is often too conservative in practice. Thus, we should search for η values that bring high empirical performances. When using Algorithm 1 as a subroutine of gradient-based methods, it is repeatedly called to solve similar equilibrium computation instances. Therefore, a simple and effective way for locating good η values is to apply Algorithm 1 with various η values to example instances, as we will do in Section 4.1. We expect the empirical performance to improve with a line search of η, which we leave for future work.

[2] Smoothness parameter L defined on $\mathbb{R}^d$ can be, in general, exponentially large in n, albeit constant in T. How to alleviate the dependence on L remains an open problem.

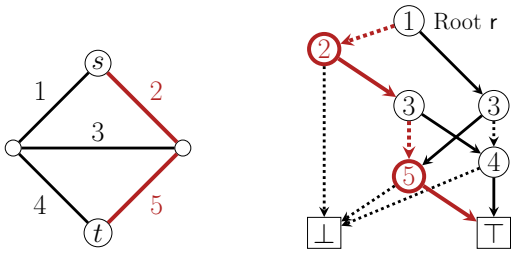


Figure 1: Example of ZDD $\mathsf{Z}_{\mathcal{S}}$ (right), where $n = 5$ and $\mathcal{S}$ is a family of all simple s–t paths (left). ZDD has two terminal nodes ($\top$ and $\bot$) and non-terminal nodes labeled by $l_{\mathsf{v}} \in [n]$. Solid (dashed) arcs represent 1-arcs (0-arcs).

Algorithm 2 Computation of $\boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c})$ with ZDD $\mathsf{Z}_{\mathcal{S}} = (\mathsf{V}, \mathsf{A})$

$\mathrm{B}_{\top} = 1$ and $\mathrm{B}_{\bot} = 0$
for $\mathsf{v} \in \mathsf{V} \setminus \{\top, \bot\}$ (bottom-up) **:**
    $\mathrm{B}_{\mathsf{v}} = \mathrm{B}_{\mathsf{c}^0_{\mathsf{v}}} + \exp(-c_{l_{\mathsf{v}}}) \times \mathrm{B}_{\mathsf{c}^1_{\mathsf{v}}}$
$\mathrm{P}_{\mathsf{r}} = 1$ and $\mathrm{P}_{\mathsf{v}} = 0$ ($\mathsf{v} \in \mathsf{V} \setminus \{\mathsf{r}\}$)
$\boldsymbol{x} = (0, \dots, 0)^{\top}$
for $\mathsf{v} \in \mathsf{V} \setminus \{\top, \bot\}$ (top-down) **:**
    $\mathrm{p}^0 = \mathrm{B}_{\mathsf{c}^0_{\mathsf{v}}} / \mathrm{B}_{\mathsf{v}}$ and $\mathrm{p}^1 = 1 - \mathrm{p}^0$
    $\mathrm{P}_{\mathsf{c}^0_{\mathsf{v}}} \mathrel{+}= \mathrm{p}^0 \mathrm{P}_{\mathsf{v}}$ and $\mathrm{P}_{\mathsf{c}^1_{\mathsf{v}}} \mathrel{+}= \mathrm{p}^1 \mathrm{P}_{\mathsf{v}}$
    $x_{l_{\mathsf{v}}} \mathrel{+}= \mathrm{p}^1 \mathrm{P}_{\mathsf{v}}$
return $\boldsymbol{x} = (x_1, \dots, x_n)^{\top}$

To check whether the Frank–Wolfe algorithm has converged or not, we usually use the Frank–Wolfe gap [27], an upper-bound on an objective error. To obtain high-quality solutions, we terminate the algorithm when the gap becomes sufficiently small. In our case, however, our purpose is to obtain gradient information $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$, and $\boldsymbol{y}_T(\boldsymbol{\theta})$ with a small T sometimes suffices to serve the purpose. By using a small T, we can reduce the computation complexity. Experiments in Section 4.2 show that the performance of a projected gradient method that uses $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$ is not so sensitive to T values.

3 Efficient softmin computation with decision diagrams

This section describes how to efficiently compute $\boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c})$ by leveraging ZDD representations of $\mathcal{S}$. The main idea is to apply a technique called weight pushing [40] (or path kernel [56]) to ZDDs. A similar idea was used for obtaining efficient combinatorial bandit algorithms [52], but this research does not use it to develop differentiable algorithms. Our use of weight pushing comes from another important observation: it consists of simple arithmetic operations that accept reverse-mode automatic differentiation with respect to $\boldsymbol{c}$, as shown in Algorithm 2. In other words, Algorithm 2 does not use, e.g., $|\cdot|$ or argmin. Therefore, ZDD-based weight pushing can be incorporated into the pipeline from $\boldsymbol{\theta}$ to $\boldsymbol{y}_T(\boldsymbol{\theta})$ without breaking the differentiability.

3.1 Zero-suppressed binary decision diagrams

Given set family $\mathcal{S} \subseteq 2^{[n]}$, we explain how to represent it with ZDD $\mathsf{Z}_{\mathcal{S}} = (\mathsf{V}, \mathsf{A})$, a DAG-shaped data structure (see, e.g., Figure 1). The node set, V, has two terminal nodes $\top$ and $\bot$ (they represent true and false, respectively) and non-terminal nodes. There is a single root, $\mathsf{r} \in \mathsf{V} \setminus \{\bot, \top\}$. Each $\mathsf{v} \in \mathsf{V} \setminus \{\bot, \top\}$ has label $l_{\mathsf{v}} \in [n]$ and two outgoing arcs, 1- and 0-arcs, which indicate whether l_{v} is chosen or not, respectively. Let $\mathsf{c}^0_{\mathsf{v}}, \mathsf{c}^1_{\mathsf{v}} \in \mathsf{V}$ denote two nodes pointed by 0- and 1-arcs, respectively, outgoing from v. For any $\mathsf{v} \in \mathsf{V} \setminus \{\top, \bot\}$, let $\mathcal{R}_{\mathsf{v}} \subseteq 2^{\mathsf{A}}$ be the set of all directed paths from v to $\top$. We define $\mathcal{R} \coloneqq \bigcup_{\mathsf{v} \in \mathsf{V} \setminus \{\top, \bot\}} \mathcal{R}_{\mathsf{v}}$. For any $\mathsf{R} \in \mathcal{R}$, let $X(\mathsf{R}) \coloneqq \left\{ l_{\mathsf{v}} \in [n] \;\middle|\; (\mathsf{v}, \mathsf{c}^1_{\mathsf{v}}) \in \mathsf{R} \right\}$, i.e., labels of tails of 1-arcs belonging to R. ZDD $\mathsf{Z}_{\mathcal{S}}$ represents $\mathcal{S}$ as a set of r–$\top$ paths: $\mathcal{S} = \{ X(\mathsf{R}) \mid \mathsf{R} \in \mathcal{R}_{\mathsf{r}} \}$. There is a one-to-one correspondence between $S \in \mathcal{S}$ and $\mathsf{R} \in \mathcal{R}_{\mathsf{r}}$, i.e., $S = X(\mathsf{R})$.

Figure 1 presents an example of ZDD $\mathsf{Z}_{\mathcal{S}}$, where $\mathcal{S}$ is the family of simple s–t paths. For example, $S = \{2, 5\} \in \mathcal{S}$ is represented in $\mathsf{Z}_{\mathcal{S}}$ by the red path, $\mathsf{R} \in \mathcal{R}_{\mathsf{r}}$, with labels $\{1, 2, 3, 5, \top\}$. The labels of the tails of the 1-arcs form $X(\mathsf{R}) = \{2, 5\}$, which equals S.

We define the size of $\mathsf{Z}_{\mathcal{S}} = (\mathsf{V}, \mathsf{A})$ by $|\mathsf{Z}_{\mathcal{S}}| \coloneqq |\mathsf{V}|$. Note that $|\mathsf{A}| \leq 2 \times |\mathsf{Z}_{\mathcal{S}}|$ always holds. In general, the ZDD sizes and the complexity of constructing ZDDs can be exponential in n. Fortunately, many existing studies provide efficient methods for constructing compact ZDDs. One such method is the frontier-based search [30], which is based on Knuth's Simpath algorithm [32]. Their method is particularly effective when $\mathcal{S}$ is a family of network substructures such as Hamiltonian paths, Steiner trees, matchings, and cliques. Furthermore, the family algebra [39, 32] of ZDDs enables us to deal with various logical constraints. Using those methods, we can flexibly construct ZDDs for various complicated combinatorial structures, e.g., Steiner trees whose size is at most a certain value.

Moreover, we can sometimes theoretically bound the ZDD sizes and the construction complexity. For example, if $\mathcal{S}$ consists of the aforementioned substructures on network $G = (V, E)$ with a constant pathwidth, the ZDD sizes and the construction complexity are polynomial in $|E|$ [30, 26].

3.2 Details of Algorithm 2 and computation complexity

Algorithm 2 computes $\boldsymbol{x} = \boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c})$ for any $\boldsymbol{c} = (c_1, \dots, c_n)^\top \in \mathbb{R}^n$. First, it computes $\{\mathrm{B}_{\mathsf{v}}\}_{\mathsf{v}\in\mathsf{V}}$ in a bottom-up topological order of $\mathsf{Z}_{\mathcal{S}}$. Note that $\mathrm{B}_{\mathsf{v}} = \sum_{S\in\{\, X(\mathsf{R}) \mid \mathsf{R}\in\mathcal{R}_{\mathsf{v}}\,\}} \exp(-\sum_{i\in S} c_i)$ holds. Then it computes $\{\mathrm{P}_{\mathsf{v}}\}_{\mathsf{v}\in\mathsf{V}}$. Each P_{v} indicates the probability that a top-down random walk starting from root node r reaches $\mathsf{v} \in \mathsf{V}$, where we choose 0-arc (1-arc) with probability p^0 (p^1). From $\mathrm{B}_{\perp} = 0$ and the construction of $\{\mathrm{B}_{\mathsf{v}}\}_{\mathsf{v}\in\mathsf{V}}$, the random walk never reaches $\perp$, and its trajectory $\mathsf{R} \in \mathcal{R}_{\mathsf{r}}$ recovers $X(\mathsf{R}) \in \mathcal{S}$ with a probability proportional to $\exp(-\sum_{i\in X(\mathsf{R})} c_i) = \exp(-\boldsymbol{c}^\top \mathbf{1}_{X(\mathsf{R})})$. Therefore, by summing the probabilities of reaching v and choosing a 1-arc outgoing from v for each $i \in [n]$ as in Step 9, we obtain $\boldsymbol{x} = \boldsymbol{\mu}_{\mathcal{S}}(\boldsymbol{c})$. In practice, we recommend implementing Algorithm 2 with the log-sum-exp technique and double-precision computations for numerical stability.

Algorithm 2 runs in $\mathrm{O}(|\mathsf{Z}_{\mathcal{S}}|)$ time, and thus Algorithm 1 takes $\mathrm{O}((n + C_\nabla + |\mathsf{Z}_{\mathcal{S}}|)T)$ time, where C_∇ is the cost of computing ∇f. From the cheap gradient principle [21], the complexity of computing $\nabla F(\boldsymbol{\theta}, \boldsymbol{y}_T(\boldsymbol{\theta}))$ with backpropagation is almost the same as that of computing $F(\boldsymbol{\theta}, \boldsymbol{y}_T(\boldsymbol{\theta}))$. That is, smaller ZDDs make the computation of both $\boldsymbol{y}_T(\boldsymbol{\theta})$ and $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$ faster. Therefore, our method significantly benefits from the empirical compactness of ZDDs. Note that we can construct ZDD $\mathsf{Z}_{\mathcal{S}}$ in a preprocessing step; once we obtain $\mathsf{Z}_{\mathcal{S}}$, we can reuse it every time $\boldsymbol{\mu}_{\mathcal{S}}(\cdot)$ is called.

4 Experiments

Section 4.1 confirms the benefit of acceleration to empirical convergence speed. Section 4.2 demonstrates the usefulness of our method via experiments on communication network design instances. Section 4.3 presents experiments with small instances to see whether our method can empirically find globally optimal $\boldsymbol{\theta}$. Due to space limitations, we present full experimental results in Appendix C.

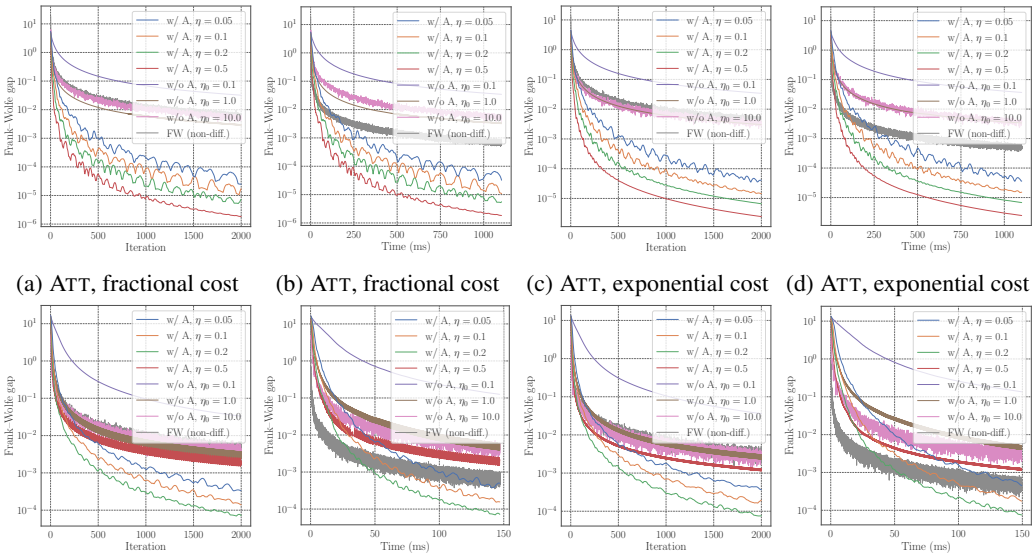
All the experiments were performed using a single thread on a 64-bit macOS machine with 2.5 GHz Intel Core i7 CPUs and 16 GB RAM. We used C++11 language, and the programs were compiled by Apple clang 12.0.0 with `-O3 -DNDEBUG` option. We used Adept 2.0.5 [23] as an automatic differentiation package and Graphillion 1.4 [25] for constructing ZDDs, where we used a beam-search-based path-width optimization method [26] to specify the traversal order of edges. The source code is available at https://github.com/nttcslab/diff-eq-comput-zdd.

Problem setting. We address Stackelberg models for optimizing network parameters $\boldsymbol{\theta} \in \mathbb{R}^n$, where $[n]$ represents an edge set. We focus on two situations where combinatorial strategies $\mathcal{S} \subseteq 2^{[n]}$ are Hamiltonian cycles and Steiner trees. The former is a variant of the selfish-routing setting, and the latter arises when designing communication networks as in Section 1.1. Note that in both settings, common operations on $\mathcal{C}$, e.g., projection and linear optimization, are NP-hard. We use two types of cost functions: fractional cost $c_i(y_i; \boldsymbol{\theta}) = d_i(1 + C \times y_i/(\theta_i + 1))$ and exponential cost $c_i(y_i; \boldsymbol{\theta}) = d_i(1 + C \times y_i \exp(-\theta_i))$, where $d_i \in (0, 1]$ is the length of the i-th edge (normalized so that $\max_{i\in[n]} d_i = 1$ holds) and $C > 0$ controls how heavily the growth in y_i (congestion) affects cost c_i. We set $C = 10$. Note that edge i with a larger θ_i is more tolerant to congestion. The leader aims to minimize social cost $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta})) = \sum_{i\in[n]} c_i(y_i(\boldsymbol{\theta}); \boldsymbol{\theta}) y_i(\boldsymbol{\theta})$. In realistic situations, the leader cannot let all edges have sufficient capacity due to budget constraints. To model this situation, we impose a constraint on $\boldsymbol{\theta}$ by defining $\Theta = \left\{ \boldsymbol{\theta} \in \mathbb{R}^n_{\geq 0} \;\middle|\; \boldsymbol{\theta}^\top \mathbf{1} = n \right\}$, where $\mathbf{1}$ is the all-one vector.

Datasets. Table 1 summarizes the information about datasets and ZDDs used in the experiments. For the Hamiltonian-cycle setting (HAMILTON), we used att48 (ATT) and dantzig42 (DANTZIG) datasets in TSPLIB [48]. Following [11, 44], we obtained networks in Figure 4 using Delaunay triangulation [54]. For the Steiner-tree setting (STEINER), we used Uninett 2011 (UNINETT) and TW Telecom (TW) networks of Internet Topology Zoo [31]. We selected terminal vertices as shown in Figure 4. We can see in Table 1 that ZDDs are much smaller than the strategy sets. As mentioned in Section 3.2, we can construct ZDDs in a preprocessing step, and the construction times were so short as to be negligible compared with the times taken for minimizing the social cost (see Section 4.2). Therefore, we do not take the construction times into account in what follows.

Table 1: Sizes of networks $G = (V, E)$, strategy sets $\mathcal{S}$, and ZDDs $\mathsf{Z}_{\mathcal{S}}$. ZDDs for DANTZIG, ATT, UNINETT, and TW were constructed in 172, 258, 4, and 6 ms, respectively.

STRATEGY	DATASET	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathsf{Z}_{\mathcal{S}}\rvert$
HAMILTON	DANTZIG	42	115	15164782028 ($\geq 1.5 \times 10^{10}$)	23479
	ATT	48	130	1041278451879 ($\geq 1.0 \times 10^{12}$)	35388
STEINER	UNINETT	69	96	88920985482584429311488 ($\geq 8.8 \times 10^{22}$)	3284
	TW	76	115	71363851011296173824385276416 ($\geq 7.1 \times 10^{28}$)	5583



(a) ATT, fractional cost (b) ATT, fractional cost (c) ATT, exponential cost (d) ATT, exponential cost

(e) TW, fractional cost (f) TW, fractional cost (g) TW, exponential cost (h) TW, exponential cost

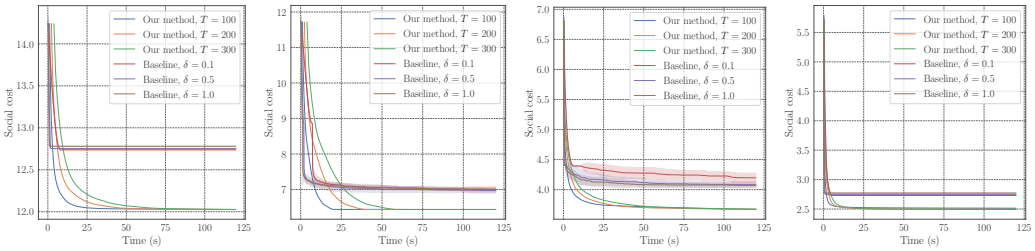
Figure 2: Convergence results of equilibrium computation methods, where w/ A, w/o A, and FW represent Algorithm 1, naive differentiable Frank–Wolfe without acceleration, and standard non-differentiable Frank–Wolfe, respectively. We present the results on the other settings in Appendix C.

4.1 Empirical convergence of equilibrium computation

We studied the empirical convergence of Algorithm 1 with acceleration (w/ A), where we let $\eta = 0.05$, 0.1, 0.2, and 0.5. We applied it to the minimization problems of form $\min_{\boldsymbol{y} \in \mathcal{C}} f(\boldsymbol{y}; \boldsymbol{\theta})$, where f is a potential function defined by cost function $c_i(y_i; \boldsymbol{\theta})$ (fractional or exponential). We let $\boldsymbol{\theta} = \mathbf{1}$.

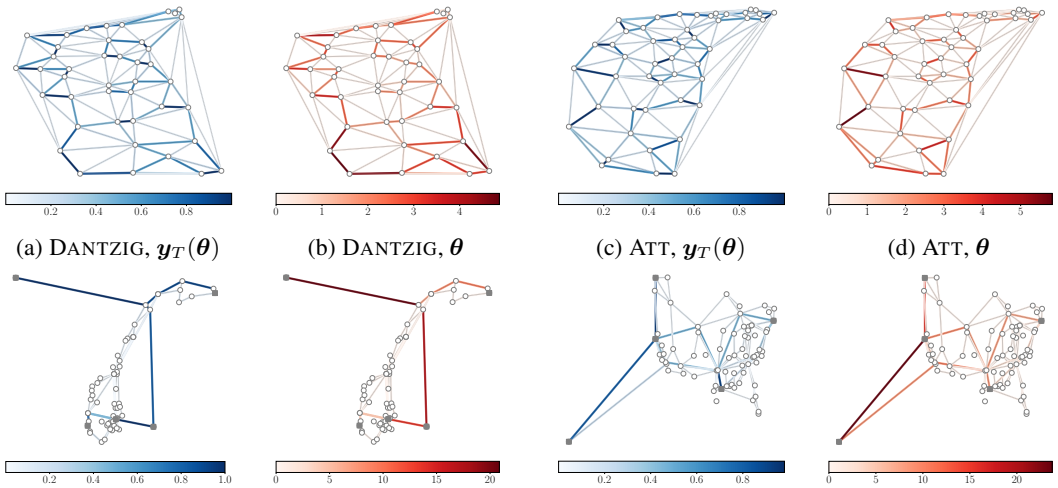
For comparison, we used two kinds of baselines. One is a differentiable Frank–Wolfe algorithm without acceleration (w/o A), which just replaces argmin with softmin as explained in Section 2.1. To guarantee the convergence of the modified algorithm, we let $\eta_t = \eta_0 \times t$ ($\eta_0 = 0.1$, 1.0, and 10.0). The other is the standard non-differentiable Frank–Wolfe algorithm (FW) implemented as in [27].

Figure 2 shows how quickly the Frank–Wolfe gap [27], which is an upper bound of the objective error, decreased as the number of iterations and the computation time increased. w/ A and w/o A tend to be faster and slower than FW, respectively. That is, Algorithm 1 (w/ A) becomes both differentiable and faster than the original FW, while the naive modified one (w/o A) becomes differentiable but slower. As in the TW results, however, w/ A with a too large η ($\eta = 0.5$) sometimes failed to be accelerated; this is reasonable since Theorem 1 requires η to be a moderate value. Thus, if we can locate appropriate η, Algorithm 1 achieves faster convergence in practice. As discussed in Section 2.4, we can search for η by examining the empirical convergence for various η values, as we did above.



(a) ATT, fractional cost (b) ATT, exponential cost (c) TW, fractional cost (d) TW, exponential cost

Figure 3: Plots of social costs achieved on network design instances. Error bands of baseline methods show standard deviations over 20 trials. We present results on other settings in Appendix C.



(a) DANTZIG, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (b) DANTZIG, $\boldsymbol{\theta}$ (c) ATT, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (d) ATT, $\boldsymbol{\theta}$

(e) UNINETT, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (f) UNINETT, $\boldsymbol{\theta}$ (g) TW, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (h) TW, $\boldsymbol{\theta}$

Figure 4: Network illustrations: five square vertices in UNINETT and TW indicate terminals. Blue (a, c, e, g) and red (b, d, f, h) edges represent $\boldsymbol{y}_T(\boldsymbol{\theta})$ and $\boldsymbol{\theta}$ values, respectively, computed by our method with $\eta = 0.1$ and $T = 300$ for fractional-cost instances.

4.2 Stackelberg models for designing communication networks

We consider minimizing social cost $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$. To this end, roughly speaking, we should assign large θ_i values to edges with large y_i values. We applied the projected gradient method with a step size of 5.0 to problem (1). We approximated $\nabla F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ by applying automatic differentiation to $F(\boldsymbol{\theta}, \boldsymbol{y}_T(\boldsymbol{\theta}))$, where $\boldsymbol{y}_T(\boldsymbol{\theta})$ was computed by Algorithm 1 with $T = 100$, 200, and 300.

To the best of our knowledge, no existing methods can efficiently deal with the problems considered here due to the complicated structures of $\mathcal{S}$. Therefore, as a baseline method, we used the following iterative heuristic. Given current $\boldsymbol{y}(\boldsymbol{\theta})$, we replace $\boldsymbol{\theta}$ with $\boldsymbol{\theta} + \delta(\boldsymbol{y}(\boldsymbol{\theta}) - \bar{y}(\boldsymbol{\theta})\mathbf{1})$, where $\delta > 0$ and $\bar{y}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i \in [n]} y_i(\boldsymbol{\theta})$. That is, we increase/decrease θ_i if edge i is used more/less than average. We then project $\boldsymbol{\theta}$ onto Θ and compute $\boldsymbol{y}(\boldsymbol{\theta})$ with Algorithm 1 ($T = 300$). If $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ value does not decrease after the above update, we restart from a random point in Θ.

Figure 3 compares our method ($\eta = 0.1$) and the baseline on ATT and TW instances, where both started from $\boldsymbol{\theta} = \mathbf{1}$ and continued to update $\boldsymbol{\theta}$ for two minutes. Our method found better $\boldsymbol{\theta}$ values than the baseline. The results only imply the empirical tendency, and our method is not guaranteed to find globally optimal $\boldsymbol{\theta}$. Nevertheless, experiments in Section 4.3 show that it tends to find a global optimum at least for small instances. Figure 4 shows the $\boldsymbol{y}_T(\boldsymbol{\theta})$ and $\boldsymbol{\theta}$ values obtained by our method with $\eta = 0.1$ and $T = 300$ for fractional-cost instance. We confirmed that large θ_i values were successfully assigned to edges with large y_i values. Those results demonstrate that our method is useful for addressing Stackelberg models of CCGs with complicated combinatorial strategy sets $\mathcal{S}$.

4.3 Experiments on empirical convergence to global optimum

We performed additional experiments on small instances to see whether the projected gradient method used in Section 4.2 can empirically find $\boldsymbol{\theta}$ that is close to being optimal. We used small selfish-routing instances, where a graph is given by the left one in Figure 1 and the edges are numbered from 1 to 5 in that order. We let strategy set $\mathcal{S}$ be the set of all simple s-t paths. The cost functions and feasible region Θ were set as with those in the above sections. Our goal is to minimize social cost $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$.

As in Section 4.2, we computed $\boldsymbol{y}(\boldsymbol{\theta})$ using Algorithm 1 with $\eta = 0.1$ and $T = 300$, and performed the projected gradient descent to minimize $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$, where gradient $\nabla F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ was computed with backpropagation. On the other hand, to obtain (approximations of) globally optimal $\boldsymbol{\theta}$, we performed an exhaustive search over the feasible region, where the step size was set to 0.05. Regarding computation times, the projected gradient method converged in less than 30 iterations, which took less than 20 ms, while the exhaustive search took about 1000 seconds.

Results on fractional costs. A global optimum found by the exhaustive search was $\boldsymbol{\theta} = (0, 2.5, 0, 0, 2.5)$, whose social cost $F(\boldsymbol{\theta}, \boldsymbol{y}(\boldsymbol{\theta}))$ was 6.444. Our method started from $\boldsymbol{\theta} = \mathbf{1}$, whose social cost was 7.000, and returned $\boldsymbol{\theta} = (1.25, 1.25, 0, 1.25, 1.25)$ with social cost 6.444. Although the solution is different from that of the exhaustive search, both attain the identical social cost. Thus, the solution returned by the projected gradient method is also globally optimal.

Results on exponential costs. A global optimum found by the exhaustive search was $\boldsymbol{\theta} = (0, 2.5, 0, 0, 2.5)$ with social cost 3.517. Our method started from $\boldsymbol{\theta} = \mathbf{1}$ with social cost 5.678 and reached $\boldsymbol{\theta} = (0, 2.5, 0, 0, 2.5)$ with social cost 3.517. Along the way, the projected gradient method was about to be trapped in $\boldsymbol{\theta} = (1.25, 1.25, 0, 1.25, 1.25)$ with social cost 4.865, which seems to be a saddle point. However, it successfully got out of there and reached the global optimum.

5 Conclusion and discussion

We proposed a differentiable pipeline that connects CCG parameters to their equilibria, enabling us to apply gradient-based methods to the Stackelberg models of CCGs. Our Algorithm 1 leverages softmin to make the Frank–Wolfe algorithm both differentiable and faster. ZDD-based softmin computation (Algorithm 2) enables us to efficiently deal with complicated CCGs. It also naturally works with automatic differentiation, offering an easy way to compute desired derivatives. Experiments confirmed the accelerated empirical convergence and practicality of our method.

An important future direction is further studying theoretical aspects. From our experimental results, $\nabla \boldsymbol{y}_T(\boldsymbol{\theta})$ is expected to converge to $\nabla \boldsymbol{y}(\boldsymbol{\theta})$, although its theoretical analysis is very difficult. Recently, some relevant results have been obtained for simple cases where iterative optimization methods are written by a contraction map defined on an unconstrained domain [1, 20]. In our CCG cases, however, we need to study iterative algorithms that numerically solve the constrained potential minimization, which requires a more profound understanding of iterative differentiation approaches. Another interesting future work is to make linearly convergent Frank–Wolfe variants [33] differentiable.

Finally, we discuss limitations and possible negative impacts. Our work does not cover cases where minimizer $\boldsymbol{y}(\boldsymbol{\theta})$ of potential functions is not unique. Since the complexity of our method mainly depends on the ZDD sizes, it does not work if ZDDs are prohibitively large, which can happen when strategy sets consist of the substructures of dense networks. Nevertheless, many real-world networks are sparse, and thus our ZDD-based method is often effective, as demonstrated in experiments. At a meta-level, optimizing social infrastructures in terms of a single objective function (e.g., the social cost) may lead to an extreme choice that is detrimental to some individuals. We hope our method can provide a basis for designing social infrastructures that are beneficial for all.

Acknowledgements

The authors thank the anonymous reviewers for their valuable feedback, corrections, and suggestions. This work was partially supported by JST ERATO Grant Number JPMJER1903 and JSPS KAKENHI Grant Number JP20H05963.

References

[1] P. Ablin, G. Peyré, and T. Moreau. Super-efficiency of automatic differentiation for functions defined as a minimum. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 32–41. PMLR, 2020.

[2] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, volume 32, pages 9562–9574. Curran Associates, Inc., 2019.

[3] Z. Allen-Zhu and L. Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference*, volume 67, pages 3:1–3:22. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[4] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 136–145. PMLR, 2017.

[5] H. Bar-Gera. Origin-based algorithm for the traffic assignment problem. *Transp. Sci.*, 36(4): 398–417, 2002.

[6] H. Bar-Gera, F. Hellman, and M. Patriksson. Computational precision of traffic equilibria sensitivities in automatic network design and road pricing. *Procedia Soc. Behav. Sci.*, 80:41–60, 2013.

[7] D. Belanger, B. Yang, and A. McCallum. End-to-end learning for structured prediction energy networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 429–439. PMLR, 2017.

[8] Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach. Learning with differentiable perturbed optimizers. In *Advances in Neural Information Processing Systems*, volume 33, pages 9508–9519. Curran Associates, Inc., 2020.

[9] U. Bhaskar, K. Ligett, L. J. Schulman, and C. Swamy. Achieving target equilibria in network routing games without knowing the latency functions. *Games Econom. Behav.*, 118:533–569, 2019.

[10] J. Bracken and J. T. McGill. Mathematical programs with optimization problems in the constraints. *Oper. Res.*, 21(1):37–44, 1973.

[11] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS J. Comput.*, 15 (3):233–248, 2003.

[12] J. R. Correa and N. E. Stier-Moses. Wardrop equilibria. In *Wiley Encyclopedia of Operations Research and Management Science*. Wiley Online Library, 2011.

[13] S. Dempe, V. Kalashnikov, G. A. Pérez-Valdés, and N. Kalashnykova. *Bilevel Programming Problems*. Springer, 1st edition, 2015.

[14] J. Domke. Generic methods for optimization-based modeling. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, volume 22, pages 318–326. PMLR, 2012.

[15] T. Fiez, B. Chasnov, and L. Ratliff. Implicit learning dynamics in Stackelberg games: Equilibria characterization, convergence analysis, and empirical study. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 3133–3144. PMLR, 2020.

[16] S. Fiorini, S. Massar, S. Pokutta, H. R. Tiwary, and R. de Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *J. ACM*, 62(2):1–23, 2015.

[17] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1568–1577. PMLR, 2018.

[18] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Res. Logis. Quart.*, 3 (1-2):95–110, 1956.

[19] S. Ghadimi and M. Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

[20] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo. On the iteration complexity of hypergradient computation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 3748–3758. PMLR, 2020.

[21] A. Griewank and A. Walther. *Evaluating Derivatives*. SIAM, 2nd edition, 2008.

[22] P. Hansen, B. Jaumard, and G. Savard. New branch-and-bound rules for linear bilevel programming. *SIAM J. Sci. Statist. Comput.*, 13(5):1194–1217, 1992.

[23] R. J. Hogan. Adept 2.0: a combined automatic differentiation and array library for C++, 2017.

[24] M. Imase and B. M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete. Math.*, 4(3): 369–384, 1991.

[25] T. Inoue, H. Iwashita, J. Kawahara, and S. Minato. Graphillion: software library for very large sets of labeled graphs. *Int. J. Software Tool. Tech. Tran.*, 18(1):57–66, 2016.

[26] Y. Inoue and S. Minato. Acceleration of ZDD construction for subgraph enumeration via path-width optimization. Technical report, TCS-TR-A-16-80, Hokkaido University, 2016.

[27] M. Jaggi. Revisiting Frank–Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 427–435. PMLR, 2013.

[28] O. Jahn, R. H. Möhring, A. S. Schulz, and N. E. Stier-Moses. System-optimal routing of traffic flows with user constraints in networks with congestion. *Oper. Res.*, 53(4):600–616, 2005.

[29] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-Softmax. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[30] J. Kawahara, T. Inoue, H. Iwashita, and S. Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E100.A(9):1773–1784, 2017.

[31] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *IEEE J. Sel. Areas Commum.*, 29(9):1765–1775, 2011. http://www.topology-zoo.org/dataset.html.

[32] D. E. Knuth. *The Art of Computer Programming: Combinatorial Algorithms, Part 1*, volume 4A. Addison-Wesley Professional, 1st edition, 2011.

[33] S. Lacoste-Julien and M. Jaggi. On the global linear convergence of Frank–Wolfe optimization variants. In *Advances in Neural Information Processing Systems*, volume 28, pages 496–504. Curran Associates, Inc., 2015.

[34] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, 1989.

[35] C. Li, H. Yang, D. Zhu, and Q. Meng. A global optimization method for continuous network design problems. *Transport. Res. B-Meth.*, 46(9):1144–1158, 2012.

[36] Z.-Q. Luo, J.-S. Pang, and D. Ralph. *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press, 1996.

[37] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2113–2122. PMLR, 2015.

[38] A. Mensch and M. Blondel. Differentiable dynamic programming for structured prediction and attention. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3462–3471. PMLR, 2018.

[39] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, pages 272–277. IEEE, 1993.

[40] M. Mohri. *Weighted Automata Algorithms*, pages 213–254. Springer, 2009.

[41] D. Monderer and L. S. Shapley. Potential games. *Games Econ. Behav.*, 14(1):124–143, 1996.

[42] K. Nakamura, S. Sakaue, and N. Yasuda. Practical Frank–Wolfe method with decision diagrams for computing Wardrop equilibrium of combinatorial congestion games. In *Processings of the 34th AAAI Conference on Artificial Intelligence*, volume 34, pages 2200–2209, 2020.

[43] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Dokl. Akad. Nauk SSSR*, 269(3):543–547, 1983.

[44] M. Nishino, N. Yasuda, S. Minato, and M. Nagata. Compiling graph substructures into sentential decision diagrams. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 1213–1221, 2017.

[45] P. Ochs, R. Ranftl, T. Brox, and T. Pock. Techniques for gradient-based bilevel optimization with non-smooth lower level problems. *J. Math. Imaging Vis.*, 56(2):175–194, 2016.

[46] M. Patriksson and R. T. Rockafellar. A mathematical model and descent algorithm for bilevel traffic management. *Transport. Sci.*, 36(3):271–291, 2002.

[47] M. V. Pogančić, A. Paulus, V. Musil, G. Martius, and M. Rolinek. Differentiation of black-box combinatorial solvers. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[48] G. Reinelt. TSPLIB—a traveling salesman problem library. *INFORMS Journal on Computing*, 3(4):376–384, 1991. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[49] R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *Int. J. Game Theory*, 2(1):65–67, 1973.

[50] T. Roughgarden. *Selfish Routing and the Price of Anarchy*. The MIT Press, 2005.

[51] S. Sakaue. Differentiable greedy algorithm for monotone submodular maximization: Guarantees, gradient estimators, and applications. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, volume 130, pages 28–36. PMLR, 2021.

[52] S. Sakaue, M. Ishihata, and S. Minato. Efficient bandit combinatorial optimization algorithm with zero-suppressed binary decision diagrams. In *Processings of the 21st International Conference on Artificial Intelligence and Statistics*, volume 84, pages 585–594. PMLR, 2018.

[53] W. H. Sandholm. Potential games with continuous player sets. *J. Econ. Theory*, 97(1):81–108, 2001.

[54] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, pages 203–222. Springer, 1996. https://www.cs.cmu.edu/~quake/triangle.html.

[55] H. Stackelberg. *The Theory of the Market Economy*. Oxford University Press, 1952.

[56] E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. *J. Mach. Learn. Res.*, 4(Oct):773–818, 2003.

[57] J. Thai. *On learning Game-Theoretical models with Application to Urban Mobility*. PhD thesis, UC Berkeley, 2017. ProQuest ID: Thai_berkeley_0028E_17598. Merritt ID: ark:/13030/m59s6nbq. Retrieved from https://escholarship.org/uc/item/3b61v84v.

[58] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1–2):1–305, 2008.

[59] J.-K. Wang and J. Abernethy. Acceleration through optimistic no-regret dynamics. In *Advances in Neural Information Processing Systems*, volume 31, pages 3824–3834. Curran Associates, Inc., 2018.

[60] B. Wilder, B. Dilkina, and M. Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

Appendix

A Extension to asymmetric CCGs

First, we introduce additional notation and definitions for extending our problem setting to the case with asymmetric CCGs, where there are multiple types of strategy sets. We can recover the simpler symmetric setting, which we studied in the main paper, by modifying the notation presented below: $r = 1$, $\mathcal{S}^1 = \mathcal{S}$, and $m^1 = 1$.

A.1 Problem setting

There are r populations of players, whose strategy sets are $\mathcal{S}^1, \dots, \mathcal{S}^r \subseteq 2^{[n]}$. Let $d^p \coloneqq |\mathcal{S}^p|$ for $p \in [r]$ and define $d \coloneqq d^1 + \cdots + d^r$. Let $\boldsymbol{z}^p \in \triangle^{d^p}$ for each $p \in [r]$ and $\boldsymbol{z} = (\boldsymbol{z}^1, \dots, \boldsymbol{z}^r) \in \mathcal{D} \coloneqq \triangle^{d^1} \times \cdots \times \triangle^{d^r}$. Let $z^p_S \in [0,1]$ be the entry of $\boldsymbol{z}^p$ corresponding to $S \in \mathcal{S}^p$, which indicates the proportion of players in the p-th group who choose strategy $S \in \mathcal{S}^p$.

For each $p \in [r]$, let $\boldsymbol{\Lambda}^p \in \{0,1\}^{n \times d^p}$ be a matrix whose (i, S) entry is 1 iff $i \in [n]$ is included in $S \in \mathcal{S}^p$; the columns of $\boldsymbol{\Lambda}^p$ consist of $\{\mathbf{1}_S\}_{S \in \mathcal{S}^p}$. Let $\boldsymbol{x}^p = \sum_{S \in \mathcal{S}^p} z^p_S \mathbf{1}_S = \boldsymbol{\Lambda}^p \boldsymbol{z}^p \in [0,1]^n$, whose i-th entry is the proportion of players in p who choose $S \in \mathcal{S}^p$ such that $i \in S$. Note that $\boldsymbol{x}^p$ is in the convex hull, $\mathrm{conv}(\mathcal{S}^p) \coloneqq \left\{ \sum_{S \in \mathcal{S}} z^p_S \mathbf{1}_S \;\middle|\; \boldsymbol{z}^p \in \triangle^{d^p} \right\}$.

For each $p \in [r]$, let $m^p > 0$ be the total mass of players in the p-th group. We define $\boldsymbol{\Lambda} \coloneqq [m^1 \boldsymbol{\Lambda}^1, \dots, m^r \boldsymbol{\Lambda}^r] \in \mathbb{R}^{n \times d}$ and let $\boldsymbol{y}$ be a vector whose i-th entry indicates the total mass of players using $i \in [n]$, i.e., $\boldsymbol{y} = \sum_{p \in [r]} m^p \boldsymbol{x}^p = \sum_{p \in [r]} m^p \boldsymbol{\Lambda}^p \boldsymbol{z}^p = \boldsymbol{\Lambda} \boldsymbol{z}$. Note that for any $\boldsymbol{z} \in \mathcal{D}$, $\boldsymbol{y} = \boldsymbol{\Lambda} \boldsymbol{z}$ is always included in $\mathcal{C} \coloneqq \left\{ \sum_{p \in [r]} m^p \boldsymbol{x}^p \;\middle|\; \boldsymbol{x}^p \in \mathrm{conv}(\mathcal{S}^p) \text{ for } p \in [r] \right\}$.

Analogous to the symmetric case, each $i \in [n]$ has cost function $c_i(\cdot; \boldsymbol{\theta})$, where we assume $c_i(y_i; \boldsymbol{\theta})$ to be strictly increasing in y_i for any $\boldsymbol{\theta} \in \Theta$ and differentiable in $\boldsymbol{\theta}$ for any $\boldsymbol{y} \in \mathcal{C}$. A player choosing strategy S incurs cost $c_S(\boldsymbol{y}; \boldsymbol{\theta}) \coloneqq \sum_{i \in S} c_i(y_i; \boldsymbol{\theta})$. In the asymmetric setting, once $\boldsymbol{\theta}$ is fixed, an equilibrium is defined as follows: $\boldsymbol{z} \in \mathcal{D}$ attains an equilibrium if for every $p \in [r]$, every $S \in \mathcal{S}^p$ such that $z^p_S > 0$ satisfies $c_S(\boldsymbol{y}; \boldsymbol{\theta}) \leq \min_{S' \in \mathcal{S}^p} c_{S'}(\boldsymbol{y}; \boldsymbol{\theta})$ for $\boldsymbol{y} = \boldsymbol{\Lambda} \boldsymbol{z}$. That is, for every $p \in [r]$, no player in the p-th group is motivated to change his/her strategy in $\mathcal{S}^p$.

As with the symmetric case, $\boldsymbol{z} \in \mathcal{D}$ attains an equilibrium iff $\boldsymbol{y} = \boldsymbol{\Lambda} \boldsymbol{z}$ is a (unique) minimizer of the following potential function minimization:

$$\underset{\boldsymbol{y}}{\text{minimize}} \;\; f(\boldsymbol{y}; \boldsymbol{\theta}) \quad \text{subject to } \boldsymbol{y} \in \mathcal{C},$$

where $f(\boldsymbol{y}; \boldsymbol{\theta}) = \sum_{i \in [n]} \int_0^{y_i} c_i(y; \boldsymbol{\theta}) \mathrm{d}y$ is a parameterized potential function. In what follows, we also consider the following formulation of the above problem:

$$\underset{\boldsymbol{z}}{\text{minimize}} \;\; \Phi(\boldsymbol{z}; \boldsymbol{\theta}) \coloneqq f(\boldsymbol{\Lambda} \boldsymbol{z}; \boldsymbol{\theta}) \quad \text{subject to } \boldsymbol{z} \in \mathcal{D}. \tag{A1}$$

Since d is exponential in n in general, we cannot directly deal with problem (A1) in practice. We only use the formulation for the theoretical analysis; our method does not explicitly deal with (A1).

A.2 Extension of our algorithm

To address the asymmetric setting, we need to slightly modify our algorithm. Algorithm A1 presents details of the modified algorithm. In Step 5, we use softmin oracle $\boldsymbol{\mu}_{\mathcal{S}^p}$ for each $p \in [r]$ to obtain $\boldsymbol{x}^p$, and in Step 6 we aggregate them to obtain $\boldsymbol{x}_t = \sum_{p \in [r]} m^p \boldsymbol{x}^p_t$. In the symmetric setting, where $r = 1$, $\mathcal{S}^1 = \mathcal{S}$, and $m^1 = 1$, Algorithm A1 is equivalent to Algorithm 1. For each $p \in [r]$, softmin oracle $\boldsymbol{\mu}_{\mathcal{S}^p}$ returns the following n-dimensional vector:

$$\boldsymbol{\mu}_{\mathcal{S}^p}(\boldsymbol{c}) = \sum_{S \in \mathcal{S}^p} \mathbf{1}_S \frac{\exp\left(-\boldsymbol{c}^\top \mathbf{1}_S\right)}{\sum_{S' \in \mathcal{S}^p} \exp\left(-\boldsymbol{c}^\top \mathbf{1}_{S'}\right)},$$

where $\boldsymbol{c} \in \mathbb{R}^n$. To compute softmin for each $p \in [r]$ with ZDDs, we construct $\mathsf{Z}_{\mathcal{S}^1}, \dots, \mathsf{Z}_{\mathcal{S}^r}$ in a preprocessing step and use Algorithm 2. As with the symmetric setting, once $\mathsf{Z}_{\mathcal{S}^p}$ is constructed for each $p \in [r]$, we can repeatedly use it every time $\boldsymbol{\mu}_{\mathcal{S}^p}$ is called.

Algorithm A1 Differentiable Frank–Wolfe-based equilibrium computation for asymmetric settings

1: $\boldsymbol{c}_0 = 0$, $\boldsymbol{s}_0 = 0$, $\boldsymbol{x}_{-1} = \boldsymbol{x}_0 = \sum_{p\in[r]} m^p \boldsymbol{\mu}_{\mathcal{S}^p}(m^p \boldsymbol{c}_0)$, and $\alpha_t = t$ $(t = 0, \dots, T)$
2: **for** $t = 1, \dots, T$ **:**
3: $\quad \boldsymbol{s}_t = \boldsymbol{s}_{t-1} - \alpha_{t-1}\boldsymbol{x}_{t-2} + (\alpha_{t-1} + \alpha_t)\boldsymbol{x}_{t-1}$
4: $\quad \boldsymbol{c}_t = \boldsymbol{c}_{t-1} + \eta\alpha_t \nabla f\left(\frac{2}{t(t+1)}\boldsymbol{s}_t; \boldsymbol{\theta}\right)$
5: $\quad \boldsymbol{x}_t^p = \boldsymbol{\mu}_{\mathcal{S}^p}(m^p \boldsymbol{c}_t)$ for each $p \in [r]$
6: $\quad \boldsymbol{x}_t = \sum_{p\in[r]} m^p \boldsymbol{x}_t^p$
return $\boldsymbol{y}_T(\boldsymbol{\theta}) = \frac{2}{T(T+1)} \sum_{t=1}^T \alpha_t \boldsymbol{x}_t$

B Proof of Theorem 1

We prove the following convergence guarantee of Algorithm A1.

Theorem A1. *If $\eta \in [\frac{1}{CL}, \frac{1}{4L}]$ holds for some constant $C > 4$, we have*

$$f(\boldsymbol{y}_T(\boldsymbol{\theta}); \boldsymbol{\theta}) - \min_{\boldsymbol{y}\in\mathcal{C}} f(\boldsymbol{y}; \boldsymbol{\theta}) \leq \mathrm{O}\left(\frac{CL\sum_{p\in[r]} \ln d^p}{T^2}\right).$$

Here L is the smoothness parameter of Φ defined in (A1). I.e., for $\Phi(\boldsymbol{z}; \boldsymbol{\theta}) \coloneqq f(\boldsymbol{\Lambda}\boldsymbol{z}; \boldsymbol{\theta})$, we assume that $\Phi(\boldsymbol{z}'; \boldsymbol{\theta}) \leq \Phi(\boldsymbol{z}; \boldsymbol{\theta}) + \langle \nabla\Phi(\boldsymbol{z}; \boldsymbol{\theta}), \boldsymbol{z}' - \boldsymbol{z}\rangle + \frac{L}{2}\|\boldsymbol{z}' - \boldsymbol{z}\|^2$ holds for all $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{R}^d$. Note that by setting $r = 1$ as explained in Appendix A, we can recover the converge guarantee of the symmetric setting (Theorem 1 in Section 2.3). Below we fix $\boldsymbol{\theta} \in \Theta$ and omit $\boldsymbol{\theta}$ for simplicity.

The following analysis is based on [59]. Our technical contribution is to reveal that their accelerated algorithm (Algorithm A2) can be used as a differentiable Frank–Wolfe algorithm (Algorithm A1), which explicitly accepts backpropagation and enjoys efficient ZDD-based implementation. Note that since Wang and Abernethy [59] did not mention the differentiability of Algorithm A2, our work is the first to show that the Frank–Wolfe algorithm can simultaneously be made differentiable and faster.

We introduce some notation and definitions. We define the Kullback–Leibler (KL) divergence as $D_{\mathrm{KL}}(\boldsymbol{z}; \boldsymbol{z}') \coloneqq \langle \boldsymbol{z}, \ln(\boldsymbol{z}/\boldsymbol{z}')\rangle$ $(\forall \boldsymbol{z}, \boldsymbol{z}' \in \mathcal{D})$, where $\boldsymbol{z}/\boldsymbol{z}'$ and $\ln$ are an element-wise division and a logarithm, respectively. Let $\alpha_t = t$ and denote the sequence of α_ts by $\boldsymbol{\alpha}_{1:t} = \alpha_1, \dots, \alpha_{t-1}, \alpha_t$. We also define modified sequence $\boldsymbol{\alpha'}_{1:t-1} = \alpha'_1, \dots, \alpha'_{t-1}$ so that $\alpha'_s = \alpha_s$ $(s \leq t-2)$ and $\alpha'_{t-1} = \alpha_{t-1} + \alpha_t$ hold, where $\boldsymbol{\alpha'}_{1:0} = \alpha_0 = 0$. Let $A_t = \sum_{s=1}^t \alpha_s$. For any sequence of vectors $\boldsymbol{u}_0, \dots, \boldsymbol{u}_t$, we define $\boldsymbol{u}_{\boldsymbol{\alpha}_{1:t}} = \sum_{s=1}^t \alpha_s \boldsymbol{u}_s$, $\bar{\boldsymbol{u}}_{\boldsymbol{\alpha}_{1:t}} = \frac{1}{A_t}\sum_{s=1}^t \alpha_s \boldsymbol{u}_s$, $\boldsymbol{u}_{\boldsymbol{\alpha'}_{1:t-1}} = \sum_{s=1}^{t-1} \alpha'_s \boldsymbol{u}_s$, and $\bar{\boldsymbol{u}}_{\boldsymbol{\alpha'}_{1:t-1}} = \frac{1}{A_t}\sum_{s=1}^{t-1} \alpha'_s \boldsymbol{u}_s$, where we let $\boldsymbol{u}_{\boldsymbol{\alpha'}_{1:0}} = \bar{\boldsymbol{u}}_{\boldsymbol{\alpha'}_{1:0}} = \boldsymbol{u}_0$.

To prove Theorem A1, we use the following relationship between Algorithms A1 and A2.

Lemma A1. *For $\boldsymbol{x}_0, \dots, \boldsymbol{x}_T$ and $\boldsymbol{z}_0, \dots, \boldsymbol{z}_T$ obtained in Step 6 of Algorithm A1 and Step 4 of Algorithm A2, respectively, we have $\boldsymbol{x}_t = \boldsymbol{\Lambda}\boldsymbol{z}_t$ $(t = 0, \dots, T)$.*

Proof of Lemma A1. We first show that for each $p \in [r]$, $\boldsymbol{z}_t^p \in \triangle^{d^p}$ obtained by Algorithm A2 satisfies

$$\boldsymbol{z}_t^p \propto \exp\left(-\sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s^p\right) \qquad (t = 1, \dots, T),$$

where $\exp$ is taken in an element-wise manner. From the KKT condition of argmin in Step 4 of Algorithm A2, for each $p \in [r]$, we have

$$\alpha_t \boldsymbol{g}_t^p + \frac{1}{\eta}(\ln \boldsymbol{z}^p + \boldsymbol{1} - \ln \boldsymbol{z}_{t-1}^p) - \nu^p \boldsymbol{1} = 0,$$

where $\nu^p \in \mathbb{R}$ is a multiplier corresponding to the equality constraint, $\boldsymbol{1}^\top \boldsymbol{z}_t^p = 1$. Note that we need not take inequality constraint $\boldsymbol{z}^p \geq 0$ into account since entropic regularization forces z_S^p to be positive. The above equality implies that entries in $\boldsymbol{z}_t^p$ are proportional to those of $\exp(-\eta\alpha_t \boldsymbol{g}_t^p + \ln \boldsymbol{z}_{t-1}^p)$, and thus we obtain $\boldsymbol{z}_t^p \propto \boldsymbol{z}_0^p \odot \exp(-\sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s^p)$ by induction, where $\odot$ is the element-wise product. Since $\boldsymbol{z}_0^p = (1/d^p, \dots, 1/d^p)$, we get $\boldsymbol{z}_t^p \propto \exp(-\sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s^p)$.

Algorithm A2 Accelerated Frank–Wolfe algorithm as a two-player game [59]

$\boldsymbol{z}_0 = (\boldsymbol{z}_0^1, \dots, \boldsymbol{z}_0^r)$ where $\boldsymbol{z}_0^p = (1/d^p, \dots, 1/d^p) \in \triangle^{d^p}$ for each $p \in [r]$
for $t = 1, \dots, T$ **:**
  $\boldsymbol{g}$-player's action: $\boldsymbol{g}_t = \nabla\Phi\left(\bar{\boldsymbol{z}}_{\boldsymbol{\alpha}'_{1:t-1}}\right)$ ▷ I.e., $\boldsymbol{g}_t = \operatorname{argmin}_{\boldsymbol{g}\in\mathbb{R}^d} \Phi^*(\boldsymbol{g}) - \langle \bar{\boldsymbol{z}}_{\boldsymbol{\alpha}'_{1:t-1}}, \boldsymbol{g}\rangle$
  $\boldsymbol{z}$-player's action: $\boldsymbol{z}_t = \operatorname{argmin}_{\boldsymbol{z}\in\mathcal{D}} \langle \alpha_t \boldsymbol{g}_t, \boldsymbol{z}\rangle + \frac{1}{\eta} D_{\mathrm{KL}}(\boldsymbol{z}; \boldsymbol{z}_{t-1})$
return $\bar{\boldsymbol{z}}_{\boldsymbol{\alpha}_{1:T}}$

We then show by induction that Algorithm A1 computes $\boldsymbol{x}_t$ that satisfies $\boldsymbol{x}_t = \boldsymbol{\Lambda}\boldsymbol{z}_t$, where $\boldsymbol{z}_t^p \propto \exp(-\sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s^p)$ holds for $t \geq 1$ as shown above. The base case of $t = 0$ can be confirmed as follows. Since $\boldsymbol{c}_0 = 0$, we have

$$\boldsymbol{\mu}_{\mathcal{S}^p}(m^p \boldsymbol{c}_0) = \sum_{S\in\mathcal{S}^p} \mathbf{1}_S \frac{\exp\left(-m^p \boldsymbol{c}_0^\top \mathbf{1}_S\right)}{\sum_{S'\in\mathcal{S}^p} \exp\left(-m^p \boldsymbol{c}_0^\top \mathbf{1}_{S'}\right)} = \sum_{S\in\mathcal{S}^p} \mathbf{1}_S \frac{1}{d^p} = \boldsymbol{\Lambda}^p \boldsymbol{z}_0^p,$$

which implies

$$\boldsymbol{x}_0 = \sum_{p\in[r]} m^p \boldsymbol{\mu}_{\mathcal{S}^p}(m^p \boldsymbol{c}_0) = \sum_{p\in[r]} m^p \boldsymbol{\Lambda}^p \boldsymbol{z}_0^p = \boldsymbol{\Lambda}\boldsymbol{z}_0.$$

We then assume that $\boldsymbol{x}_s = \boldsymbol{\Lambda}\boldsymbol{z}_s$ holds for $s = 0, \dots, t-1$. In the t-th step, Algorithm A1 computes

$$\boldsymbol{s}_t = \boldsymbol{x}_{\boldsymbol{\alpha}'_{1:t-1}} \qquad \text{and} \qquad \boldsymbol{c}_t = \boldsymbol{c}_0 + \sum_{s=1}^t \eta\alpha_s \nabla f(\bar{\boldsymbol{x}}_{\boldsymbol{\alpha}'_{1:s-1}}) = \sum_{s=1}^t \eta\alpha_s \nabla f(\bar{\boldsymbol{x}}_{\boldsymbol{\alpha}'_{1:s-1}}).$$

From the induction hypothesis, it holds that $\frac{2}{t(t+1)}\boldsymbol{s}_t = \bar{\boldsymbol{x}}_{\boldsymbol{\alpha}'_{1:t-1}} = \boldsymbol{\Lambda}\bar{\boldsymbol{z}}_{\boldsymbol{\alpha}'_{1:t-1}}$, which implies

$$\nabla\Phi(\bar{\boldsymbol{z}}_{\boldsymbol{\alpha}'_{1:s-1}}) = \boldsymbol{\Lambda}^\top \nabla f(\bar{\boldsymbol{x}}_{\boldsymbol{\alpha}'_{1:s-1}}).$$

With these equations, we obtain

$$\boldsymbol{\Lambda}^\top \boldsymbol{c}_t = \sum_{s=1}^t \eta\alpha_s \boldsymbol{\Lambda}^\top \nabla f(\bar{\boldsymbol{x}}_{\boldsymbol{\alpha}'_{1:s-1}}) = \sum_{s=1}^t \eta\alpha_s \nabla\Phi(\bar{\boldsymbol{z}}_{\boldsymbol{\alpha}'_{1:s-1}}) = \sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s,$$

and thus $m^p \boldsymbol{\Lambda}^{p\top} \boldsymbol{c}_t = \sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s^p$ holds for each $p \in [r]$. Therefore, for each $p \in [r]$, $\boldsymbol{x}_t^p$ computed in Algorithm A1 satisfies

$$\boldsymbol{x}_t^p = \boldsymbol{\mu}(m^p \boldsymbol{\Lambda}^{p\top} \boldsymbol{c}_t) = \boldsymbol{\Lambda}^p \frac{\exp\left(-m^p \boldsymbol{\Lambda}^{p\top} \boldsymbol{c}_t\right)}{Z_t^p} = \boldsymbol{\Lambda}^p \frac{\exp\left(-\sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s^p\right)}{Z_t^p} = \boldsymbol{\Lambda}^p \boldsymbol{z}_t^p,$$

where Z_t^p is a normalizing constant and the last equality uses $\boldsymbol{z}_t^p \propto \exp(-\sum_{s=1}^t \eta\alpha_s \boldsymbol{g}_s^p)$. Hence we obtain $\boldsymbol{x}_t = \sum_{p\in[r]} m^p \boldsymbol{x}_t^p = \sum_{p\in[r]} m^p \boldsymbol{\Lambda}^p \boldsymbol{z}_t^p = \boldsymbol{\Lambda}\boldsymbol{z}_t$. Consequently, the lemma holds by induction. □

Owing to Lemma A1, we can analyze the convergence of Algorithm A1 through Algorithm A2. We regard Algorithm A2 as the dynamics of a two-player zero-sum game, where $\boldsymbol{g}$-player computes $\boldsymbol{g}_t$ and $\boldsymbol{z}$-player computes $\boldsymbol{z}_t$. The payoff function of the game is a convex-linear function defined as $u(\boldsymbol{g}, \boldsymbol{z}) \coloneqq \Phi^*(\boldsymbol{g}) - \langle \boldsymbol{g}, \boldsymbol{z}\rangle$, where $\Phi^*(\boldsymbol{g}) \coloneqq \sup_{\boldsymbol{z}\in\mathbb{R}^d}\{\langle \boldsymbol{g}, \boldsymbol{z}\rangle - \Phi(\boldsymbol{z})\}$ is the Fenchel conjugate of Φ. As detailed in [59], the players' actions are given by online optimization algorithms (in particular, $\boldsymbol{g}$-player uses a so-called *optimistic* online algorithm), and the *regret* analysis of the online algorithms yields an accelerated convergence guarantee. Formally, the following lemma holds.

Lemma A2 ([59])**.** *Algorithm A2 returns* $\bar{\boldsymbol{z}}_{\boldsymbol{\alpha}_{1:T}}$ *satisfying* $\Phi(\bar{\boldsymbol{z}}_{\boldsymbol{\alpha}_{1:T}}) - \min_{\boldsymbol{z}\in\mathcal{D}} \Phi(\boldsymbol{z}) \leq \frac{2CLB}{T(T+1)}$, *where* $B = D_{\mathrm{KL}}(\boldsymbol{z}^*; \boldsymbol{z}_0)$ *and* $\boldsymbol{z}^* \in \operatorname{argmin}_{\boldsymbol{z}\in\mathcal{D}} \Phi(\boldsymbol{z})$.

The proof of Lemma A2 is presented in [59, Theorem 2 and Corollary 1], where $\boldsymbol{z}$-player's action is described using Bregman divergence instead of KL divergence. Since KL divergence is a special case of Bregman divergence defined with convex function $\psi(\boldsymbol{u}) = \langle \boldsymbol{u}, \ln \boldsymbol{u}\rangle$ ($\boldsymbol{u} \in \mathcal{D}$), which is 1-strongly

convex over $\mathcal{D}$, we can directly apply their result to our setting. Moreover, in their analysis, the step size is given by a non-increasing sequence, $\eta_1, \ldots, \eta_T$, to obtain a more general result. We here use a simplified version such that $\eta_1 = \cdots = \eta_T = \eta$. In this case, $B = D_{\mathrm{KL}}(\boldsymbol{z}^*; \boldsymbol{z}_0)$ appears as a leading factor as in Lemma A2 (see [59, Lemma 4] for details).

By using Lemmas A1 and A2, we can obtain Theorem A1 as follows.

Proof of Theorem A1. Note that we have $\boldsymbol{y}_T(\boldsymbol{\theta}) = \frac{2}{T(T+1)} \sum_{t=1}^{T} \alpha_t \boldsymbol{x}_t = \bar{\boldsymbol{y}}_{\boldsymbol{\alpha}_{1:T}}$. From Lemma A1, we have $\boldsymbol{x}_t = \boldsymbol{\Lambda} \boldsymbol{z}_t$, where $\boldsymbol{x}_t$ and $\boldsymbol{z}_t$ are those computed in Algorithms A1 and A2, respectively. Thus, we have $\boldsymbol{y}_T(\boldsymbol{\theta}) = \bar{\boldsymbol{y}}_{\boldsymbol{\alpha}_{1:T}} = \boldsymbol{\Lambda} \bar{\boldsymbol{z}}_{\boldsymbol{\alpha}_{1:T}}$. The convergence of Algorithm A2 is guaranteed by Lemma A2. Moreover, we have $B = D_{\mathrm{KL}}(\boldsymbol{z}^*; \boldsymbol{z}_0) \leq \sum_{p \in [r]} \ln d^p$ since $\boldsymbol{z}_0^p = (1/d^p, \ldots, 1/d^p) \in \triangle^{d^p}$ holds for each $p \in [r]$. Therefore, from $f(\boldsymbol{y}) = \Phi(\boldsymbol{z})$ for $\boldsymbol{y} = \boldsymbol{\Lambda} \boldsymbol{z}$, we obtain Theorem A1. □

C Full version of experiments

We present full versions of the experimental results.

C.1 Empirical convergence of equilibrium computation

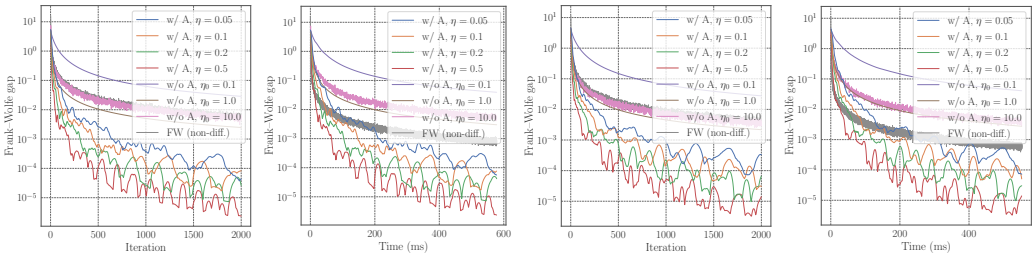
We compared the empirical convergence of three algorithms: our algorithm with acceleration (w/ A), the differentiable Frank–Wolfe algorithm without acceleration (w/o A), and the standard Frank–Wolfe algorithm (FW). We used potential minimization problems, $\min_{\boldsymbol{y} \in \mathcal{C}} f(\boldsymbol{y}; \boldsymbol{\theta})$, detailed in Section 4.1.

Figure 5 shows the results. Similar to those in Section 4.1, the performance of the differentiable Frank–Wolfe algorithm with acceleration (w/ A) tends to exceed the others (w/o A and FW), although w/ A with $\eta = 0.5$ sometimes fails to be accelerated due to the too large η value.

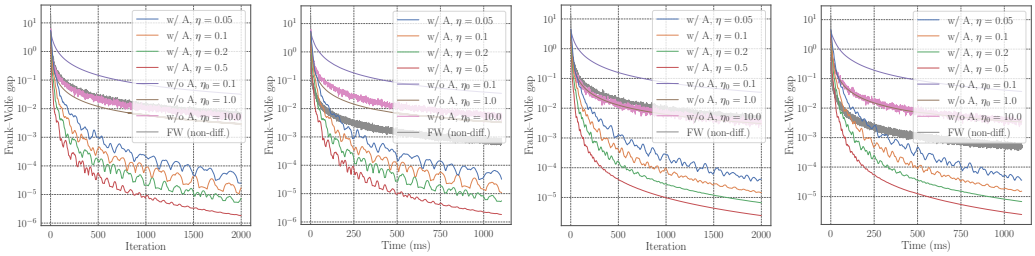
C.2 Stackelberg model for designing communication networks

We present full versions of the results on the Stackelberg model experiments described in Section 4.2.

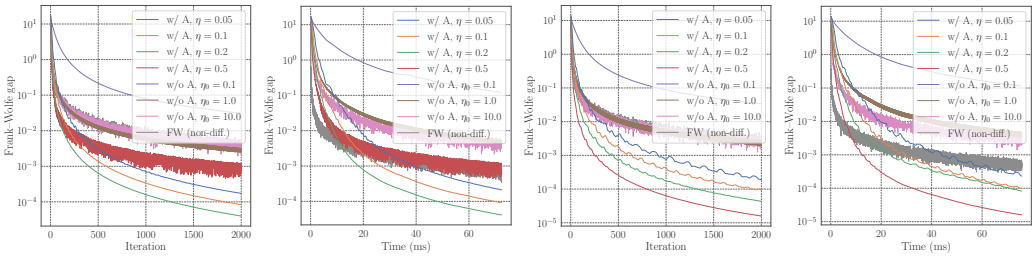
In Figures 6 and 7, we present the social-cost results. Our method outperformed the baseline in every setting. Figures 8 and 9 present full versions of the $(\boldsymbol{y}_T(\boldsymbol{\theta}), \boldsymbol{\theta})$ illustrations. We can see that our method successfully assigned large θ_i values to the edges with large $y_i(\boldsymbol{\theta})$ values.



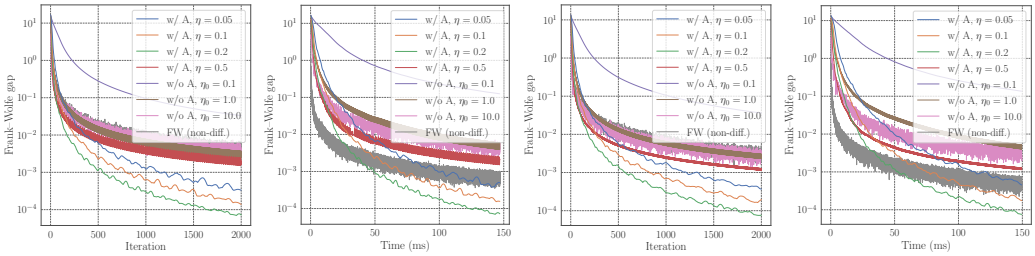
(a) DANTZIG, fractional (b) DANTZIG, fractional (c) DANTZIG, exponential (d) DANTZIG, exponential



(e) ATT, fractional (f) ATT, fractional (g) ATT, exponential (h) ATT, exponential

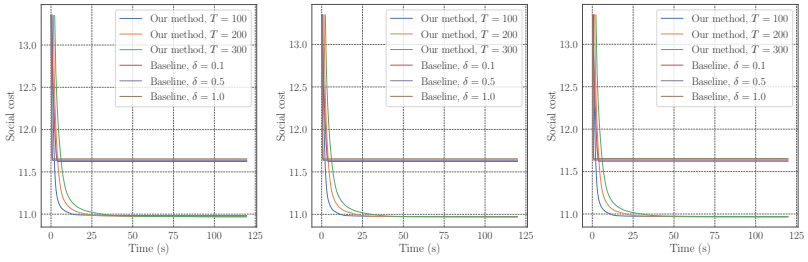


(i) UNINETT, fractional (j) UNINETT, fractional (k) UNINETT, exponential (l) UNINETT, exponential

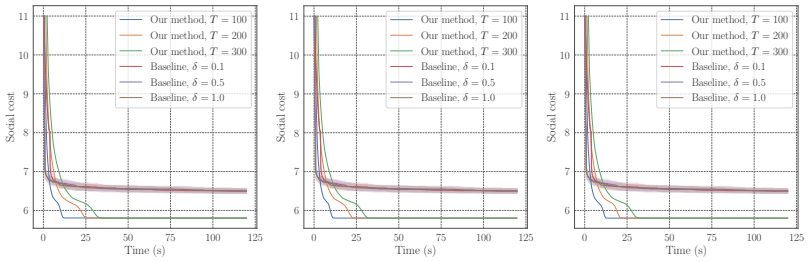


(m) TW, fractional (n) TW, fractional (o) TW, exponential (p) TW, exponential

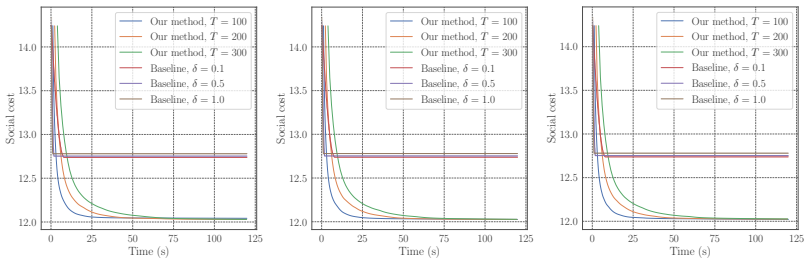
Figure 5: Convergence results on DANTZIG, ATT, UNINETT, and TW instances with fractional and exponential costs.



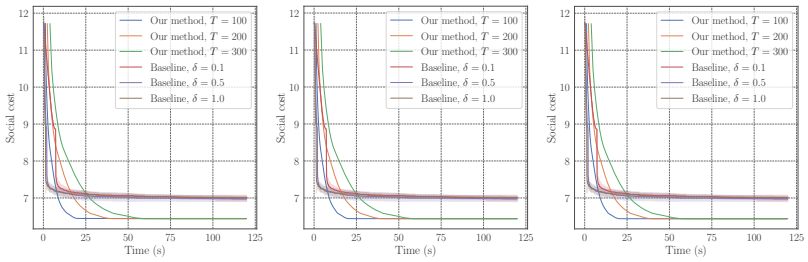
(a) DANTZIG, F, $\eta = 0.05$ (b) DANTZIG, F, $\eta = 0.1$ (c) DANTZIG, F, $\eta = 0.2$



(d) DANTZIG, E, $\eta = 0.05$ (e) DANTZIG, E, $\eta = 0.1$ (f) DANTZIG, E, $\eta = 0.2$

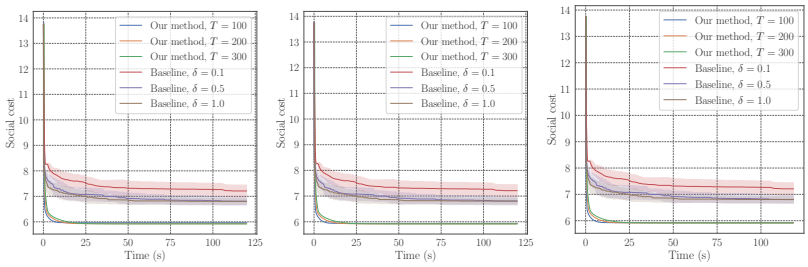


(g) ATT, F, $\eta = 0.05$ (h) ATT, F, $\eta = 0.1$ (i) ATT, F, $\eta = 0.2$

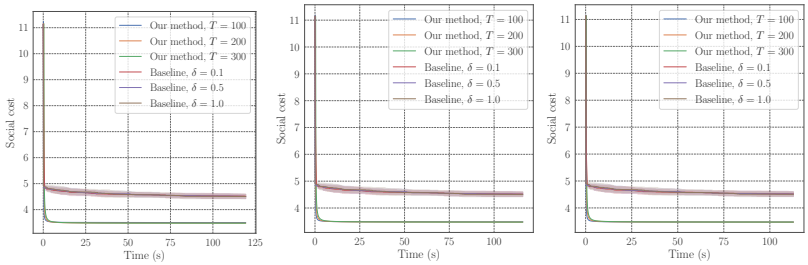


(j) ATT, E, $\eta = 0.05$ (k) ATT, E, $\eta = 0.1$ (l) ATT, E, $\eta = 0.2$

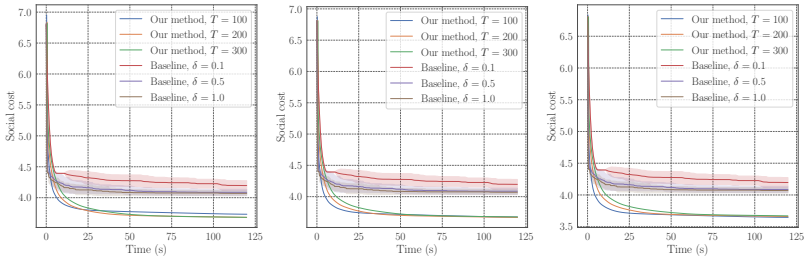
Figure 6: Plots of social costs for DANTZIG and ATT instances with fractional (F) and exponential (E) costs. We set η of Algorithm 1 to 0.05, 0.1, and 0.2. Baseline method results are shown with means and standard deviations over 20 random trials.



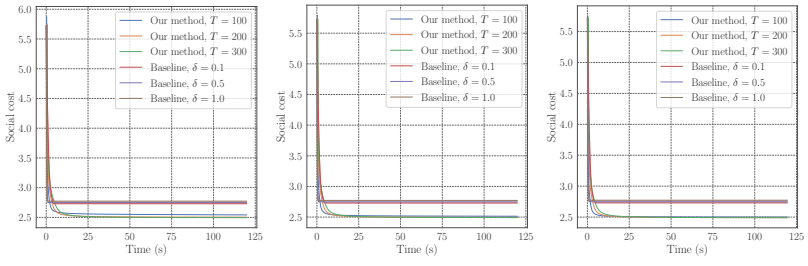
(a) UNINETT, F, $\eta = 0.05$ (b) UNINETT, F, $\eta = 0.1$ (c) UNINETT, F, $\eta = 0.2$



(d) UNINETT, E, $\eta = 0.05$ (e) UNINETT, E, $\eta = 0.1$ (f) UNINETT, E, $\eta = 0.2$

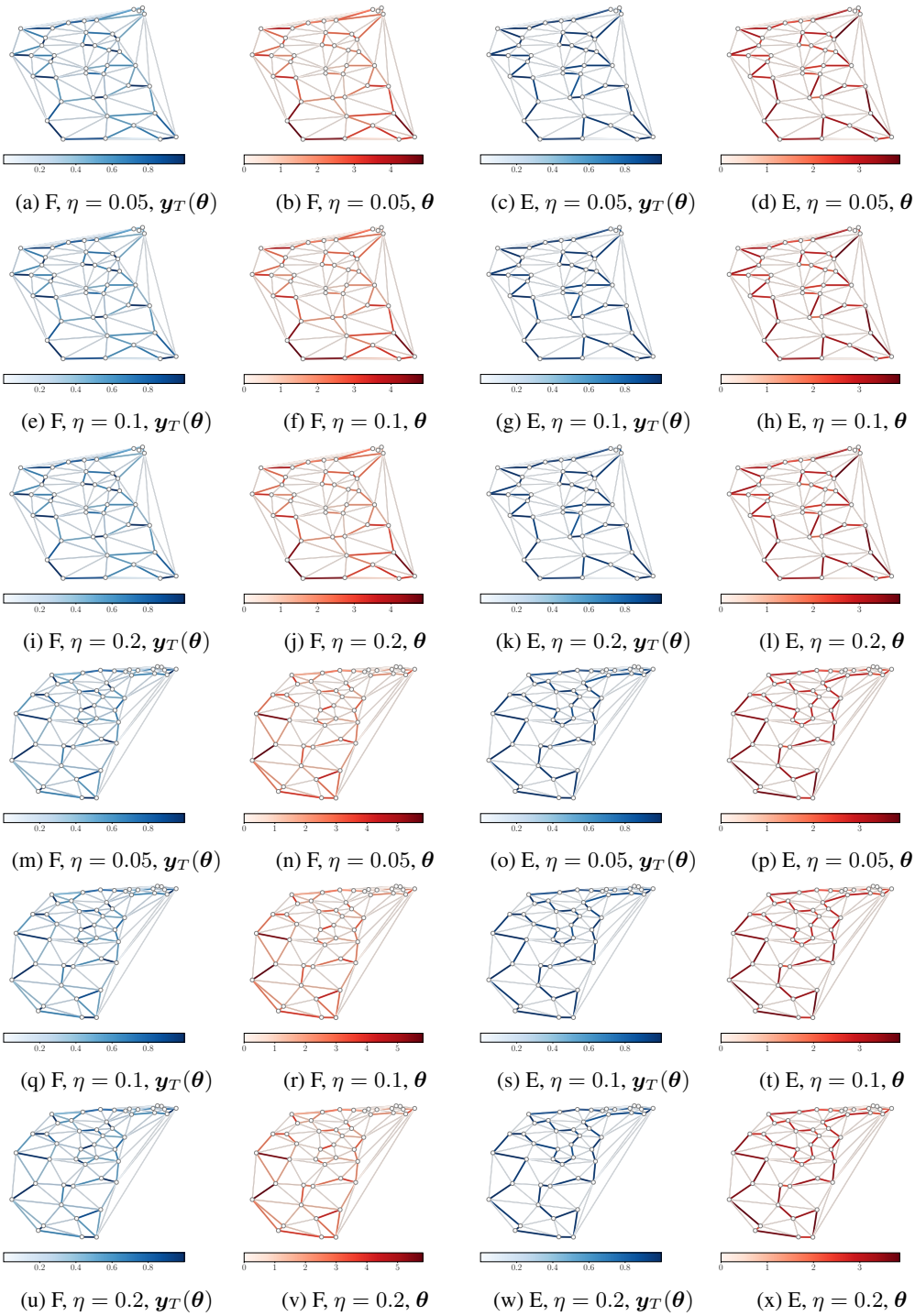


(g) TW, F, $\eta = 0.05$ (h) TW, F, $\eta = 0.1$ (i) TW, F, $\eta = 0.2$



(j) TW, E, $\eta = 0.05$ (k) TW, E, $\eta = 0.1$ (l) TW, E, $\eta = 0.2$

Figure 7: Plots of social costs for UNINETT and TW instances with fractional (F) and exponential (E) costs. We set η of Algorithm 1 to 0.05, 0.1, and 0.2. Baseline method results are shown with means and standard deviations over 20 random trials.



(a) F, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (b) F, $\eta = 0.05$, $\boldsymbol{\theta}$ (c) E, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (d) E, $\eta = 0.05$, $\boldsymbol{\theta}$

(e) F, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (f) F, $\eta = 0.1$, $\boldsymbol{\theta}$ (g) E, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (h) E, $\eta = 0.1$, $\boldsymbol{\theta}$

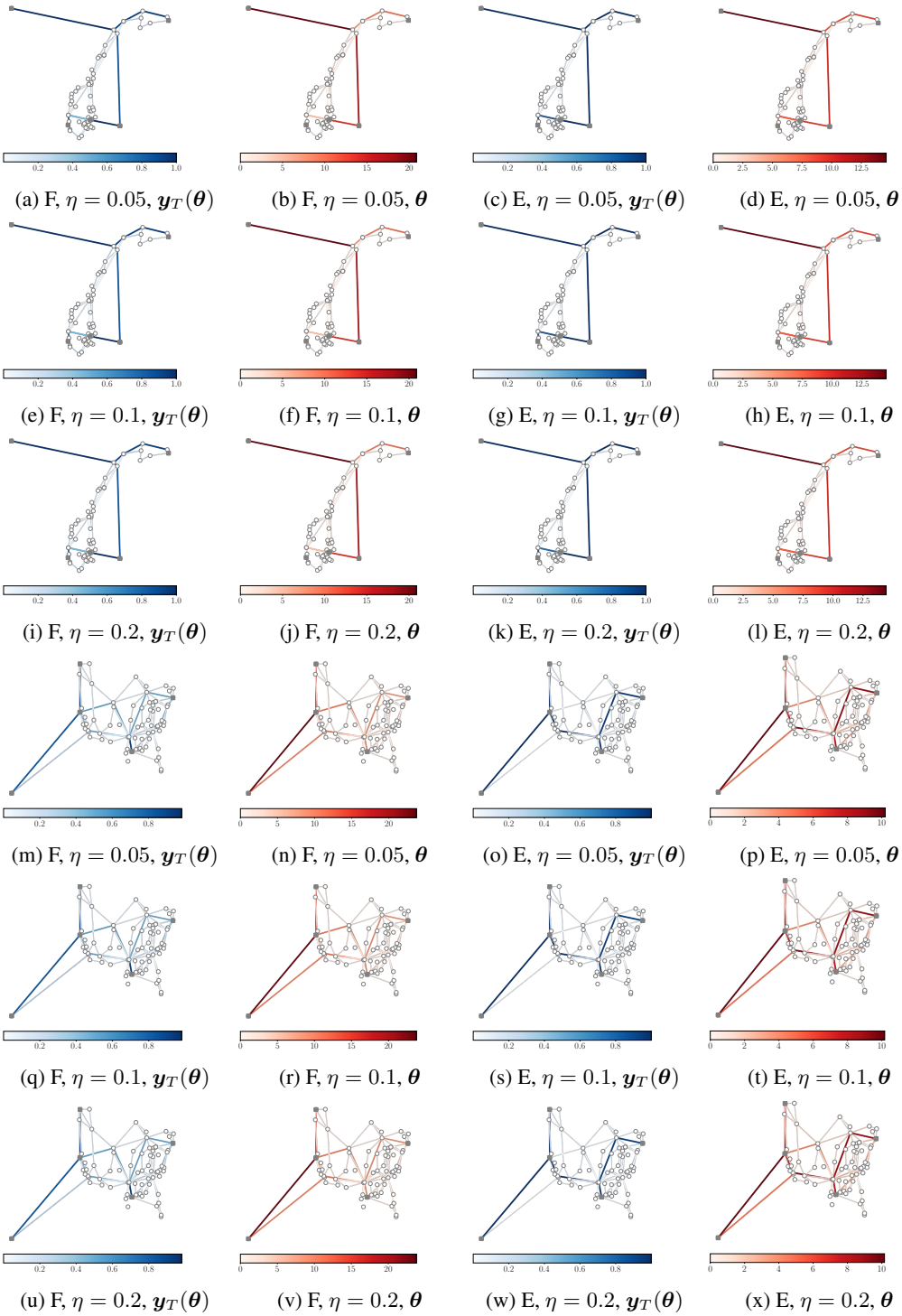
(i) F, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (j) F, $\eta = 0.2$, $\boldsymbol{\theta}$ (k) E, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (l) E, $\eta = 0.2$, $\boldsymbol{\theta}$

(m) F, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (n) F, $\eta = 0.05$, $\boldsymbol{\theta}$ (o) E, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (p) E, $\eta = 0.05$, $\boldsymbol{\theta}$

(q) F, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (r) F, $\eta = 0.1$, $\boldsymbol{\theta}$ (s) E, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (t) E, $\eta = 0.1$, $\boldsymbol{\theta}$

(u) F, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (v) F, $\eta = 0.2$, $\boldsymbol{\theta}$ (w) E, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (x) E, $\eta = 0.2$, $\boldsymbol{\theta}$

Figure 8: Illustration of DANTZIG (a–l) and ATT (m–x) networks. Blue and red edges represent $\boldsymbol{y}_T(\boldsymbol{\theta})$ and $\boldsymbol{\theta}$ values computed by our method with $T = 300$ and $\eta = 0.05, 0.1, 0.2$ for fractional (F) and exponential (E) cost instances.



(a) F, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (b) F, $\eta = 0.05$, $\boldsymbol{\theta}$ (c) E, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (d) E, $\eta = 0.05$, $\boldsymbol{\theta}$

(e) F, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (f) F, $\eta = 0.1$, $\boldsymbol{\theta}$ (g) E, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (h) E, $\eta = 0.1$, $\boldsymbol{\theta}$

(i) F, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (j) F, $\eta = 0.2$, $\boldsymbol{\theta}$ (k) E, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (l) E, $\eta = 0.2$, $\boldsymbol{\theta}$

(m) F, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (n) F, $\eta = 0.05$, $\boldsymbol{\theta}$ (o) E, $\eta = 0.05$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (p) E, $\eta = 0.05$, $\boldsymbol{\theta}$

(q) F, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (r) F, $\eta = 0.1$, $\boldsymbol{\theta}$ (s) E, $\eta = 0.1$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (t) E, $\eta = 0.1$, $\boldsymbol{\theta}$

(u) F, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (v) F, $\eta = 0.2$, $\boldsymbol{\theta}$ (w) E, $\eta = 0.2$, $\boldsymbol{y}_T(\boldsymbol{\theta})$ (x) E, $\eta = 0.2$, $\boldsymbol{\theta}$

Figure 9: Illustration of UNINETT (a–l) and TW (m–x) networks. Five square vertices in each network are terminals. Blue and red edges represent $\boldsymbol{y}_T(\boldsymbol{\theta})$ and $\boldsymbol{\theta}$ values computed by our method with $T = 300$ and $\eta = 0.05, 0.1, 0.2$ for fractional (F) and exponential (E) cost instances.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 1.1 and the statement of Theorem 1.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the [Yes] See the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A] We did not train ML models.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In our experiments, only the baseline method has randomness, whose error bands are indicated in figures.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, [Yes] See Section 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] All assets are conventionally used as open-license datasets, and their sources do not explicitly mention it.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant [N/A]