

Centralized Network Utility Maximization With Accelerated Gradient Method

Ying Tian¹, Zhiliang Wang¹, Member, IEEE, Xia Yin, Senior Member, IEEE, Xingang Shi¹, Member, IEEE, Jiahai Yang¹, Senior Member, IEEE, and Han Zhang

Abstract—Network utility maximization (NUM) is a fundamental problem for network traffic management and resource allocation. Due to the inherent decentralization and complexity of networks, much of the existing research has focused on developing decentralized algorithms for NUM. However, with the rise of Software-Defined Networking (SDN), especially in cloud networks and inter-datacenter networks managed by large enterprises, there has been growing interest in centralized NUM algorithms. To cope with the large and increasing number of flows in such SDN networks, existing studies on centralized NUM focus on the scalability of the algorithm with respect to the number of flows, but the efficiency is ignored. In this paper, we propose a centralized, efficient and scalable algorithm for the NUM problem. By designing smooth utility and penalty functions, we formulate the NUM problem with a smooth objective function, which enables the use of Nesterov’s accelerated gradient method (AGM). We prove that the proposed method achieves an $O(d/t^2)$ convergence rate, demonstrating superior convergence speed with respect to the number of iterations t , and our method is scalable with respect to the number of flows d in the network. Our smooth objective NUM formulation and AGM are effective not only in simple network scenarios with non-prioritized flows routed on one simple paths, but also in more complex and practical scenarios involving prioritized flows routed across multiple complex paths. Experiment results confirm that our method obtains accurate solutions with fewer iterations, and achieves close-to-optimal network utility.

Index Terms—Network utility maximization, resource allocation, accelerated gradient method.

I. INTRODUCTION

THE rapid growth in network traffic and the number of network users has brought unprecedented challenges to effective and fair network resource allocation. The *network utility maximization* (NUM) problem, originally proposed by Kelly [1], provides a fundamental framework for addressing this challenge. In the NUM problem, multiple sources send

elastic traffic along predetermined paths, and each source has a continuously differentiable, non-decreasing and concave utility function related to its sending rate. The most commonly used utility function is the α -fair function proposed by Mo and Walrand [2], [3]. The objective of the problem is to maximize the total utility across all sources, subject to link capacity constraints. Due to its generality and flexibility, NUM has been applied across various network scenarios, including wireless network [4], multi-layer radio access network [5], network function virtualization [6] and so on.

Given the decentralized and complex nature of networks, most of the NUM research [7], [8], [9] assumes that information like link capacity and flow route are only locally known by network devices [10], and develops decentralized algorithm based on dual decomposition [11]. The NUM problem is solved iteratively in a decentralized manner, where network devices solve their subproblems locally and communicate necessary information like intermediate results. Decentralized methods usually scale with the number of concurrent flows d , and the fastest convergence rate achieved by existing decentralized methods is $O(1/t)$ [12], [13] where t represents the number of iterations.

Over the past decades, Software Defined Networking (SDN) [14] has garnered great attention because of its ability to decouple control plane from data plane, offering a centralized approach to network management. Unlike traditional decentralized networks, the SDN architecture features a central controller with a global view of the network topology and states. This SDN controller centrally computes forwarding paths, flow rates, etc., and then controls the behavior of switches in the data plane. SDN has been widely adopted, especially in cloud networks and inter-datacenter networks managed by large enterprises [15], [16], where a large number of flows are competing for scarce and expensive network resources like link bandwidth. To meet the urgent need for managing and optimizing resource allocation in such networks, it is essential to develop *centralized, efficient and scalable* algorithms for the NUM problem: the algorithm should be *centralized* to solve the problem on the SDN controller with global information, be *efficient* to achieve near-optimal results fast or even in real time, and be *scalable* to accommodate the large and even increasing number of network flows.

Existing studies about centralized NUM algorithm mainly focus on achieving scalability, but the efficiency remains sub-optimal. Gupta et al. [17] address scalability by converting the original NUM problem into a simpler problem involving only aggregate flows. However, this approach does not guarantee a fast convergence rate or high efficiency. The state-of-the-art method Exp-NUM [18] solves the penalty-based NUM

Received 22 August 2024; revised 12 January 2025; accepted 15 April 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor R. Pedarsani. Date of publication 2 May 2025; date of current version 17 October 2025. This work was supported by the National Natural Science Foundation of China under Grant 62221003. (Corresponding author: Zhiliang Wang.)

Ying Tian is with Zhongguancun Laboratory, Beijing 100094, China (e-mail: tianying@zgclab.edu.cn).

Zhiliang Wang, Xingang Shi, Jiahai Yang, and Han Zhang are with Zhongguancun Laboratory, Beijing 100094, China, and also with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: wzl@cernet.edu.cn; shixg@cernet.edu.cn; yang@cernet.edu.cn; zhhan@mail.tsinghua.edu.cn).

Xia Yin is with Zhongguancun Laboratory, Beijing 100094, China, and also with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: yxia@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TON.2025.3562283

formulation with exponentiated gradient descent, which is proven to be scalable with respect to the number of flows. Despite this, it suffers from poor efficiency due to its slow convergence rate with respect to the number of iterations t ; this method requires a large number of iterations to reach an accurate solution.

In this paper, we follow the classical formulation of the NUM problem with d concurrent flows and the α -fair utility function, and derive a centralized *accelerated gradient method* (AGM) for the NUM problem, which is both efficient and scalable. Unlike existing approaches that only rely on the Lipschitz continuous property of the α -fair function, we go a step further by revealing the smoothness of the α -fair function and design an appropriate smooth penalty function for the link capacity constraints. Our formulation enables the use of Nesterov's accelerated gradient method [19], which is widely considered the fastest iterative algorithm for smooth objective functions. Our method is proven to have a fast convergence rate with respect to the number of iterations t , ensuring high efficiency, while also being scalable with respect to the number of flows d . These characteristics show its great potential for resource allocation in large-scale centrally controlled networks.

The main contributions of our paper are as follows:

- We are the first to formulate the NUM problem with a smooth objective function. First, we propose and prove that the α -fair utility function becomes smooth with only minor changes. Then, we design a smooth penalty function for the penalty-based formulation, which leads to the overall smoothness of the objective function in cooperation with the smooth α -fair utility function or any other smooth utility functions. The NUM formulation with smooth objective function benefits existing algorithms, enabling the convergence rate of the vanilla gradient descent to improve from $O(1/\sqrt{t})$ to $O(1/t)$, and facilitates the design of faster algorithms.
- We propose to apply Nesterov's accelerated gradient method (AGM) to solve the centralized NUM problem. Thanks to the smoothness of the objective function, the algorithm has an $O(d/t^2)$ convergence rate under the assumption that the number of network links is much smaller than d , indicating its rapid convergence with respect to t . The iteration complexity of our algorithm is proportional to $\sqrt{d}$, so the algorithm is scalable with respect to the number of flows and can maintain good performance as d increases. Moreover, AGM is guaranteed to have a lower error upper bound than the state-of-the-art method Exp-NUM with enough number of iterations. We also use two restart heuristics to boost the convergence of AGM and alleviate the bumps during the iteration.
- We conduct extensive experiments to evaluate the algorithms and demonstrate the efficiency and scalability of AGM. The experimental results are in good agreement with the theoretical results. AGM has the fastest convergence rate compared to gradient descent and Exp-NUM, and achieves accurate solutions. Besides, our method obtains close-to-optimal utility function value, indicating the effectiveness of the smoothly penalized NUM formulation. And our method performs consistently well with different number of flows, showing its good scalability with respect to d .
- In addition to the simple NUM problem with non-prioritized flows each routing on one simple path, we

extend our approach to more complex network scenarios where prioritized flows route on multiple complex paths. We propose the smooth penalty-based formulation of the generalized NUM problem, prove the smoothness of the generalized objective function, and discuss the effectiveness of AGM. We also conduct experiments to demonstrate the consistent fast convergence and scalability of AGM for the generalized NUM problem.

The rest of the paper is structured as follows. Section II is the related work. Section III presents the formulation of the NUM problem. In Section IV, we introduce the state-of-the-art gradient based methods for centralized NUM problem. We formulate the NUM problem with smooth objective function in Sec V, and then present the accelerated gradient method in Sec VI. In section VII, we extend the simple NUM problem to consider complex network scenarios. We evaluate the proposed algorithm with the simple NUM problem and the generalized NUM problem in section VIII and section IX respectively. In section X, we discuss the limitations of our method. Finally, we make a conclusion in Section XI.

II. RELATED WORK

In the past two decades, there has been plenty of research in NUM. Most studies focus on concave utility functions, and some specifically use α -fair function in the problem formulation or as a representative example [5], [7], [8], [12], [13], [17], [18], [22]. There is also research on non-concave utility functions. For example, Ashour et al. [23] tackled the non-concave NUM problem by applying a series of convex relaxations. NUM algorithms are usually iterative, and their performance is related to: i) *convergence rate*, i.e., the achieved error ε^1 at step t , or equivalently, *iteration complexity*, i.e., the number of iterations required to achieve an error; ii) *per-iteration cost*, measured by the computational complexity or number of mathematical operations per iteration.

Newton's method is believed to have the fastest convergence rate. Wei et al. [24] proposed a distributed Newton's method for the NUM problem. Zargham et al. [25], [26] accelerated dual descent with an approximate Newton-like algorithm. However, the convergence rate is not concisely derived. Besides, Newton's method requires computing Hessian matrix and its inverse, causing heavy per-iteration cost.

Therefore, first-order methods that only require the computation of the first derivative (gradient) are preferred. Kelly et al. [7] decomposed the NUM problem into a user subproblem and a network subproblem and solve the problem with gradient descent. Low et al. [8] proposed another decomposition framework and consider asynchronous situation. Both works leverage the dual-based gradient descent algorithm, but concise proof of the convergence rate is not given. In [20], authors proved that the dual gradient method with primal averaging has $O(1/\sqrt{t})$ convergence rate. In [12], authors assumed that constraint functions are Lipschitz continuous and convex, and proposed a virtual queue algorithm based on dual gradient method, achieving $O(1/t)$ convergence rate. Assuming strong concavity of the utility function, Beck et al. [13] solved the dual problem with accelerated gradient method. But their primal-dual method only achieves $O(1/t)$ convergence rate for the primal solution.

¹The error ε is measured by the gap between the objective value of the current solution and the optimal value.

TABLE I
NETWORK TYPE, OBJECTIVE FUNCTION CLASS, CONVERGENCE RATE AND ITERATION COMPLEXITY OF THE FIRST-ORDER METHODS

Network Type	Method	Objective Function Class ^a	Convergence Rate	Iteration Complexity
Decentralized controlled	Dual gradient [20]	concave	$O(1/\sqrt{t})$	$\Omega(1/\varepsilon^2)$
	Virtual queue [12]	concave (and Lipschitz continuous constraint)	$O(1/t)$	$\Omega(1/\varepsilon)$
	Primal-dual [13]	strongly concave	$O(1/t)$	$\Omega(1/\varepsilon)$
SDN network	Projected gradient ascent ^b [21]	concave, Lipschitz continuous	$O(\sqrt{d}/\sqrt{t})$	$\Omega(d/\varepsilon^2)$
		concave, smooth	$O(d/t)$	$\Omega(d/\varepsilon)$
	Exp-NUM [18]	concave, Lipschitz continuous	$O(\sqrt{\ln d}/\sqrt{t})$	$\Omega(\ln d/\varepsilon^2)$
	AGM (this paper)	concave, smooth	$O(d/t^2)$	$\Omega(\sqrt{d}/\sqrt{\varepsilon})$

^a In this table, we assume that the NUM problem is a maximizing problem so all objective functions should be concave.

^b Projected gradient ascent has different convergence rate and iteration complexity for Lipschitz continuous and smooth objective functions.

The works mentioned above all develop distributed algorithms for decentralized controlled networks through dual decomposition. As the network architecture evolves, NUM algorithms considering other network architectures emerge. Karakoç et al. [5] proposed a multi-layer decomposition for NUM problem in the radio access network solved by dual gradient descent method. They also studied the NUM problem for mobile edge computing with a federated edge network architecture [27]. Allybokus et al. [22] considered an SDN network with multiple distributed controllers, and developed a distributed algorithm based on alternating direction method of multipliers (ADMM) algorithm.

For the NUM problem in centralized SDN networks, a classic method is projected gradient ascent [21] (or or projected gradient descent for minimizing problems). For Lipschitz continuous objective functions, it has $O(\sqrt{d}/\sqrt{t})$ convergence rate and $\Omega(d/\varepsilon^2)$ iteration complexity. For smooth objective functions, its performance guarantee is improved to $O(d/t)$ convergence rate and $\Omega(d/\varepsilon)$ iteration complexity. Because the iteration complexity of projected gradient ascent is proportional to d , it is not scalable with respect to the number of flows. To cope with the increasing number of flows, Gupta et al. [17] proposed a NUM decomposition in which the set of flows are grouped by source-destination pairs, and solved the problem with ADMM algorithm. But the convergence rate is not studied. Exp-NUM [18] is most relevant to our paper. Exp-NUM uses a penalty-based formulation of the NUM problem and additionally adds a global flow rate bound, which allows the problem to be solved with exponentiated gradient descent, i.e., mirror descent [28]. The advantage of Exp-NUM is that the iteration complexity is $\Omega(\ln d/\varepsilon^2)$, thus the algorithm is scalable with respect to the number of flows d . However, the convergence rate of Exp-NUM is $O(\sqrt{\ln d}/\sqrt{t})$, indicating that it cannot converge to accurate solutions efficiently.

Recent research explores the use of reinforcement learning (RL) to solve the NUM problem. Chen et al. [29] propose an RL algorithm with a modified reward function to address the α -fair NUM problem among multiple RL agents, and prove that the algorithm can converge to a stationary point of the NUM problem. Hao et al. [30] leverage a similar modified RL algorithm in the computing offloading problem, guaranteeing the user fairness. Compared to traditional second-order and first-order methods, RL algorithms are better suited to dynamically changing environments. However, the convergence rate of the RL algorithms for the NUM problem remains unsolved.

Our method AGM [31] achieves an $O(d/t^2)$ convergence rate by leveraging a smooth optimization objective, thereby

TABLE II
NOTATIONS

Symbol	Description
$G = (V, E)$	The network graph, where V is the node set and E is the unidirectional link set.
c_e	The capacity of link e .
S	The set of traffic flows.
d	The number of flows, i.e., $d = S $.
$\mathbf{x}$	The vector of flow rates where x_s is the rate of flow s .
$\mathbf{x}^{(t)}$	The obtained solution at iteration t .
$\mathbf{x}^*$	The optimal solution.
$\mathbf{A}$	The routing matrix where $\mathbf{A}_{e,s}$ represents whether flow s passes link e or not.
U_s, U	The flow utility function and the network utility function, where $U = \sum_{s \in S} U_s$.
P_e, P	The link penalty function and the network penalty function, where $P = \sum_{e \in E} P_e$.
V	The objective function.
V^*	The optimal objective function value.
$\mathbb{R}_+^d$	The space of d -dimensional nonnegative column vector.
$[\cdot]_+$	The projection on $\mathbb{R}_+^d$.
$\ \cdot\ $	The Euclidean norm.
$\odot$	The Hadamard product.

exhibiting an $O(1/t^2)$ dependence on t . Besides, the iteration complexity is $\Omega(\sqrt{d}/\sqrt{\varepsilon})$ in the worst case, i.e., the number of iterations required for an ε -optimal solution is proportional to the square root of the number of flows. Thus, our method is both efficient and scalable. In Table I, we summarized the network type, objective function class, convergence rate and iteration complexity of the first-order methods.

III. NETWORK UTILITY MAXIMIZATION PROBLEM

In this section, we present the formulation of the NUM problem and introduce the commonly used utility function and penalty function. The notations used in this paper is summarized in Table II.

A. Problem Formulation

The system model of the NUM problem consists of the following aspects:

- 1) *Network*: The network is represented by a graph $G = (V, E)$, where V is the router set and E is the unidirectional link set. Each link $e \in E$ has a capacity c_e .
- 2) *Flows*: Let S represents the set of elastic flows whose sending rate can be controlled by the source or controller. $s \in S$ represents a specific flow, and x_s is the rate of s . We also use $\mathbf{x} = (x_s)_{s \in S}$ as the column vector of flow rates. Besides, let d be the number of flows in the network, i.e., $d = |S|$, $\mathbf{x} \in \mathbb{R}_+^d$.
- 3) *Routing path*: The routing path is predetermined for each flow $s \in S$. We assume that there is one path for each flow and the $|E| \times |S|$ routing matrix $\mathbf{A}$ is defined as:

$$\mathbf{A}_{e,s} = \begin{cases} 1, & \text{if flow } s \text{ passes through link } e \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

This definition is suitable for simple paths along which a flow s fully passes each link. For convenience, we use the row vector $\mathbf{A}_e$ to denote the e -th row of matrix $\mathbf{A}$.

- 4) *Utility function*: Each flow s is associated with a flow utility function $U_s : \mathbb{R}_+ \rightarrow \mathbb{R}$, i.e., each flow s gains a utility $U_s(x_s)$ when it sends data at rate x_s . We call $U(\mathbf{x}) = \sum_{s \in S} U_s(x_s)$ the *network utility function*. The NUM problem is formulated as:

$$\begin{aligned} \max \quad & U(\mathbf{x}) = \sum_{s \in S} U_s(x_s) \\ \text{s.t.,} \quad & \mathbf{A}_e \mathbf{x} \leq c_e, \forall e \in E \\ & x_s \geq 0, \forall s \in S \end{aligned} \quad (\text{NUM})$$

The objective is to maximize the network utility function. It is constrained that the traffic carried by a link should not exceed its capacity, and the flow rate should be nonnegative. Usually, the utility function is defined by service providers in order to achieve a certain type of resource allocation. The α -fair function [3] is a well-known and most commonly used flow utility function, defined as Eq. (2). The α -fair function is flexible to capture different optimization goals, and different values of the parameter α yield different fairness criteria. For example, $\alpha = 0$ leads to the classic throughput maximization, $\alpha = 1$ yields proportional fairness [1], $\alpha = 2$ yields to potential delay fairness [32] and $\alpha \rightarrow \infty$ gives min-max fairness.

$$U_\alpha(x_s) = \begin{cases} \frac{x_s^{1-\alpha}}{1-\alpha}, & \alpha \geq 0, \alpha \neq 1 \\ \ln x_s, & \alpha = 1 \end{cases} \quad (2)$$

For the sake of our proposed method AGM and its associated theorems and proofs, we make the following assumptions:

Assumption 1: The network topology is sparse enough so that $|E| \ll |V|(|V| - 1)$. This assumption holds when network topologies are not full-mesh and are much sparser (e.g., the topologies in Topology Zoo dataset [33], or GÉANT and RF1221 used in VIII Evaluation).

Assumption 2: The number of flows d is at least the same order of magnitude as the number of node pairs $|V|(|V| - 1)$. d could be greater than $|V|(|V| - 1)$ because we consider fine-grained flows sent by network users rather than aggregate flows between source-destination pairs.

Assumption 3: The utility function is concave and smooth. The α -fair function is well-known to be concave, and we will reveal its smoothness later in the paper.

B. Penalty-Based Formulation

In problem (NUM), the link capacity constraints form a polyhedron and is hard to handle. For gradient based methods, projection onto the polyhedron can be time consuming. One approach to address this issue is to derive the Lagrange dual of (NUM), which is suitable for decentralized algorithms. Another conventional approach is to relax the problem by using a penalty-based formulation [10], and the objective function becomes:

$$V(\mathbf{x}) = \sum_{s \in S} U_s(x_s) - \mu \sum_{e \in E} P_e(\mathbf{A}_e \mathbf{x})$$

The function $P_e : \mathbb{R}_+ \rightarrow \mathbb{R}$ is a *link penalty function* that penalizes the arriving rate for overshooting the link capacity. We call $P(\mathbf{x}) = \sum_{e \in E} P_e(\mathbf{A}_e \mathbf{x})$ the *network penalty function*. $\mu > 0$ is a penalty parameter that balances the utility function and penalty function. To follow the convention of using gradient descent (instead of ascent) based algorithms for minimization (instead of maximization) problems, we negate the function $V(\mathbf{x})$, and then the problem is formulated as:

$$\begin{aligned} \min \quad & V_P(\mathbf{x}) = - \sum_{s \in S} U_s(x_s) + \mu \sum_{e \in E} P_e(\mathbf{A}_e \mathbf{x}) \\ \text{s.t.,} \quad & x_s \geq 0, \forall s \in S \end{aligned} \quad (\text{NUM-P})$$

The optimization objective now is to minimize the negative of the network utility functions plus the network penalty function. The advantage of the problem (NUM-P) is that the link capacity constraints are removed so that the feasible region now becomes the nonnegative orthant $\mathbb{R}_+^d$ instead of a polyhedron, making the problem easier to deal with. Besides, by using an appropriate penalty function and parameter μ , the exact value of the optimal solution of (NUM) can be obtained through solving (NUM-P) [10], [18].

A typical choice for the link penalty function is the exact penalty function [34] $P_{e,+}(\mathbf{A}_e \mathbf{x})$ as shown in Eq. (3). The penalty value is equal to how much the link capacity is exceeded. However, this function is not continuously differentiable at $\mathbf{A}_e \mathbf{x} = c_e$, which may cause numerical issues and make the problem hard to handle.

$$P_{e,+}(\mathbf{A}_e \mathbf{x}) = \max\{\mathbf{A}_e \mathbf{x} - c_e, 0\} \quad (3)$$

IV. GRADIENT METHODS FOR NUM WITH THE EXACT PENALTY FUNCTION

In this section, we introduce the classic projected gradient descent (PGD) [21] and the state-of-the-art method Exp-NUM [18] for the NUM problem with the α -fair utility function U_α (Eq. (2)) and exact penalty function $P_{e,+}$ (Eq. (3)). We describe the prerequisites, algorithm procedures, performance guarantees of these methods, and analyze their drawbacks.

Algorithm 1 Projected Gradient Descent (PGD)

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T
Output: final solution $\hat{\mathbf{x}}^{(T)}$

- 1: **for** $t \leftarrow 0$ to $T - 1$ **do**
 - 2: $\mathbf{x}^{(t+1)} \leftarrow [\mathbf{x}^{(t)} - \gamma \nabla V_P(\mathbf{x}^{(t)})]_+$
 - 3: **end for**
 - 4: $\hat{\mathbf{x}}^{(T)} \leftarrow \frac{1}{T} \sum_{t=1}^T \mathbf{x}^{(t)}$
 - 5: **return:** $\hat{\mathbf{x}}^{(T)}$
-

The pseudocode of PGD is shown in Algorithm 1. At each iteration, PGD takes a step in the opposite direction of the gradient, and then projects the obtained point back onto $\mathbb{R}_+^d$. The final solution $\hat{\mathbf{x}}^{(T)}$ is the average over all intermediate points in the iteration. To give the convergence rate of PGD, an assumption is made:

Assumption 4: The partial derivatives of $V_P(\mathbf{x})$ are bounded, i.e., there exists $M \geq 0$ so that

$$\left| \frac{\partial V_P(\mathbf{x})}{\partial x_s} \right| \leq M, \quad \forall \mathbf{x} \in \mathbb{R}_+^d, s \in S,$$

implying that $V_P(\mathbf{x})$ is $(M\sqrt{d})$ -Lipschitz continuous.²

Then PGD achieves the following convergence guarantee:

Proposition 1 (Convergence rate of PGD [21]): If Algorithm 1 is run for T iterations with initial point $\mathbf{x}^{(0)}$, step size $\gamma = \frac{\|\mathbf{x}^* - \mathbf{x}^{(0)}\|}{M\sqrt{d}}$, then the solution $\hat{\mathbf{x}}^{(T)}$ satisfies:

$$V_P(\hat{\mathbf{x}}^{(T)}) - V_P^* \leq \frac{M\sqrt{d} \|\mathbf{x}^* - \mathbf{x}^{(0)}\|}{\sqrt{T}},$$

i.e., the error is $\varepsilon = O(\sqrt{d}/\sqrt{t})$ at step t , or equivalently, PGD achieves an ε -optimal solution with $\Omega(d/\varepsilon^2)$ iterations.

According to Proposition 1, PGD is neither efficient nor scalable: numerous iterations are required to reach an accurate solution because of the slow convergence rate with respect to t , and the iterative complexity grows proportionally to the number of network flows d .

Exp-NUM [18] is the state-of-the-art gradient method for the NUM problem. Considering problem (NUM-P) with the α -fair utility function U_α and exact penalty function $P_{e,+}$, authors of Exp-NUM propose an exponentiated gradient descent method whose iteration complexity is nearly independent of the number of flows. In Exp-NUM, aside from Assumption 4, an additional assumption is made:

Assumption 5: The sum of flow rates is bounded by a parameter C , thus the feasible region of (NUM-P) becomes $\mathcal{X} = \{\mathbf{x} \in \mathbb{R}_+^d : x_s \geq 0 \text{ and } \sum_{s \in S} x_s \leq C\}$, where $C \geq 0$.

Algorithm 2 Exponentiated Gradient Descent (Exp-NUM)

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T , global flow rate bound C

Output: final solution $\hat{\mathbf{x}}^{(T)}$

```

1: for  $t \leftarrow 0$  to  $T - 1$  do
2:    $\mathbf{v}^{(t)} \leftarrow \nabla V_P(\mathbf{x}^{(t)})$ 
3:    $\mathbf{x}^{(t+1)} \leftarrow C \frac{\mathbf{x}^{(t)} \odot \exp(-\gamma \mathbf{v}^{(t)})}{C + \sum_{s \in S} x_s^{(t)} [\exp(-\gamma v^{(t)}) - 1]}$ 
4: end for
5:  $\hat{\mathbf{x}}^{(T)} \leftarrow \frac{1}{T} \sum_{t=1}^T \mathbf{x}^{(t)}$ 
6: return:  $\hat{\mathbf{x}}^{(T)}$ 
```

The pseudocode of Exp-NUM are shown in Algorithm 2. At each iteration, it updates $\mathbf{x}^{(t+1)}$ with an exponential function of the gradient. The final solution is still the averaged point $\hat{\mathbf{x}}^{(T)}$. Exp-NUM has the following convergence rate guarantee:

Proposition 2 (Convergence rate of Exp-NUM [18]): If Algorithm 2 is run for T iterations with initial point $\mathbf{x}^{(0)}$ and

step size $\gamma = M^{-1} \sqrt{2D(\mathbf{x}^*, \mathbf{x}^{(0)})}/t$, then the solution $\hat{\mathbf{x}}^{(T)}$ satisfies:

$$V_P(\hat{\mathbf{x}}^{(T)}) - V_P^* \leq \frac{M \sqrt{2CD(\mathbf{x}^*, \mathbf{x}^{(0)})}}{\sqrt{T}}, \quad (4)$$

where $D(\mathbf{x}^*, \mathbf{x}^{(0)})$ is the Bregman divergence [18] between $\mathbf{x}^*$ and $\mathbf{x}^{(0)}$. If $\mathbf{x}^{(0)}$ is the barycenter of $\mathcal{X}$, then $D(\mathbf{x}^*, \mathbf{x}^{(0)})$ is less than $C \ln(d+1)$, and ExpNUM achieves $\varepsilon = O(\sqrt{\ln d}/\sqrt{t})$ error at step t , or equivalently, Exp-NUM achieves an ε -optimal solution with $\Omega(\ln d/\varepsilon^2)$ iterations.

There are a few things to note about PGD and Exp-NUM:

- 1) Neither PGD nor Exp-NUM is efficient, and they both require a lot of iterations to converge. The advantage of Exp-NUM over PGD is that the iteration complexity is in logarithmic time relative to d , thus the algorithm is scalable with respect to the number of flows. However, its convergence rate is still proportional to $1/\sqrt{t}$.
- 2) PGD and Exp-NUM use the average point $\hat{\mathbf{x}}^{(T)}$ as the final solution, instead of the point at the last iteration $\mathbf{x}^{(T)}$. The reason is that the intermediate solutions at each iteration may oscillate around the optimal solution and every-step progress cannot be guaranteed, while using the average point yields a provable convergence guarantee.
- 3) Assumption 4 and Assumption 5 are not valid, so the convergence analyses of PGD and Exp-NUM, i.e., Proposition 1 and Proposition 2, are not solid. Assumption 4 does not hold for $V_P(\mathbf{x})$ with the α -fair utility function U_α and exact penalty function $P_{e,+}$. First, $P_{e,+}(\mathbf{A}_e \mathbf{x})$ is non-differential when $\mathbf{A}_e \mathbf{x} = c_e$, so $\frac{\partial V_P(\mathbf{x})}{\partial x_s}$ cannot be calculated. Second, even if it is artificially specified that $\frac{\partial P_{e,+}(\mathbf{A}_e \mathbf{x})}{\partial x_s} = 0$ when $\mathbf{A}_e \mathbf{x} = c_e$, there is still no upper bound M on $\left| \frac{\partial V_P(\mathbf{x})}{\partial x_s} \right|$. By Eq. (2), $U'_s(x_s) = x_s^{-\alpha}$. So when $x_s = 0$, $U'_s(x_s) = \infty$, and $\left| \frac{\partial V_P(\mathbf{x})}{\partial x_s} \right| = \infty$. Thus the partial derivatives of $V_P(\mathbf{x})$ over $\mathbf{x} \in \mathbb{R}_+^d$ (or $\mathbf{x} \in \mathcal{X}$) are not bounded.

Exp-NUM chooses to add a small number to the argument x_s of $U_\alpha(x_s)$ to handle this problem in the experiments, but no theoretical analysis is given. (See the next section for detail.) Assumption 5 alters the feasible region of (NUM-P). The optimal solution of (NUM-P) is not changed only if $C \geq \sum_{e \in E} c_e$.

V. NUM PROBLEM WITH SMOOTH OBJECTIVE FUNCTION

In this section, we propose the NUM problem formulation with smooth objective function. We begin with preliminary definitions and lemmas. We then design a smooth utility function and a smooth penalty function respectively to construct the smooth objective function. The resulting NUM problem benefits the use of PGD and Exp-NUM algorithms. More importantly, it allows for the use of the accelerated gradient method presented in the next section.

A. Preliminaries

We define the concept *Lipschitz continuous* and *smooth* with respect to the Euclidean norm $\|\cdot\|$, and introduce some relevant properties. The proof of the lemmas is trivial and straightforward, so it is omitted.

²See Definition 1 and Lemma 1 in Section V-A.

Definition 1 (Lipschitz continuous): A function f is L -Lipschitz continuous (or L -Lipschitz for short) on $Q \in \mathbb{R}^n$ with Lipschitz constant $L \geq 0$, if for all $\mathbf{x}, \mathbf{y} \in Q$, we have

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$$

Lemma 1: A function f is L -Lipschitz on $Q \in \mathbb{R}^n$ if and only if there exists $L \geq 0$ so that $\|\nabla f(\mathbf{x})\| \leq L$ for all $\mathbf{x} \in Q$.

Lemma 2: The composition of Lipschitz functions is Lipschitz. Specifically, if a function $f : X \rightarrow Y$ is L_f -Lipschitz on X , and a function $g : Y \rightarrow Z$ is L_g -Lipschitz on Y , then the function $g \circ f : X \rightarrow Z$ is $(L_f L_g)$ -Lipschitz on X .

Definition 2 (Smooth): A function f is β -smooth on $Q \in \mathbb{R}^n$ with smooth constant $\beta \geq 0$, if f has β -Lipschitz continuous gradient, i.e., for all $\mathbf{x}, \mathbf{y} \in Q$, we have

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq \beta\|\mathbf{x} - \mathbf{y}\| \quad (5)$$

Lemma 3: The sum of smooth functions is smooth. Specifically, if a function $f : X \rightarrow Y$ is β_f -smooth on X , and a function $g : X \rightarrow Y$ is β_g -smooth on X , then the function $(f + g) : X \rightarrow Y$ is $(\beta_f + \beta_g)$ -smooth on X .

B. Smooth Utility Function

According to the analysis at the end of Section IV and Lemma 1, the α -fair function is not Lipschitz continuous. Its corresponding network utility function $U(\mathbf{x}) = \sum_{s \in S} U_{\alpha}(x_s)$ is not smooth, either. $\|\nabla U(\mathbf{x}) - \nabla U(\mathbf{y})\| = \sqrt{\sum_{s \in S} (x_s^{-\alpha} - y_s^{-\alpha})^2}$. If $x_s = 0$ for some $s \in S$, then $x_s^{-\alpha} = \infty$ and $\|\nabla U(\mathbf{x}) - \nabla U(\mathbf{y})\| = \infty$. Thus, Eq. (5) does not hold.

Clearly, the “infinity” problem occurs when some x_s approaches 0. At $x_s = 0$, the derivative of the α -fair function becomes ∞ , which may invalidate certain mathematical assumptions or introduce numerical difficulties. This is a common issue faced by NUM research. For instance, in [5], the authors address it by restricting the domain of the α -fair function to $x_s > 0$, allowing them to establish an $O(1/t)$ convergence rate for their primal-dual algorithm.

In this paper, we maintain the domain of the flow utility function as $x_s \geq 0$, and introduce the (α, ξ) -fair function as a new link utility function, which is defined as:

$$U_{\alpha, \xi}(x_s) = \begin{cases} \frac{(x_s + \xi)^{1-\alpha}}{1-\alpha}, & \alpha \geq 0, \alpha \neq 1 \\ \ln(x_s + \xi), & \alpha = 1, \end{cases} \quad (6)$$

where ξ is a small number and $\xi > 0$. The formulation of the (α, ξ) -fair function is very similar to the α -fair function, except that a parameter ξ is added to x_s . Actually, it is common practice to add a small number to the argument x_s of the α -fair function to avoid numerical issues when $x_s \rightarrow 0$. For example, Exp-NUM [18] and Primal-dual [13] employed this approach in their experiments. However, existing works regard the function $U_{\alpha, \xi}(x_s)$ as a “mathematical trick” rather than incorporating it into problem formulations, and there is no theoretical analysis of the mathematical property of $U_{\alpha, \xi}(x_s)$. Here we show that it yields a smooth network utility function.

Lemma 4: The function $U(\mathbf{x}) = \sum_{s \in S} U_{\alpha, \xi}(x_s)$ is $\frac{\alpha}{\xi^{\alpha+1}}$ -smooth on $\mathbb{R}_+^d$.

Proof: First we define function $\varphi : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ as:

$$\varphi(x) = (x + \xi)^{-\alpha},$$

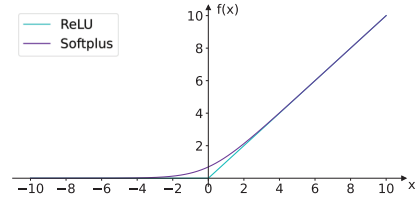


Fig. 1. Graphs of ReLU function and Softplus function.

where $\alpha \geq 0, \xi > 0$. Then we have:

$$|\varphi'(x)| = \left| \frac{\alpha}{(x + \xi)^{\alpha+1}} \right| \leq \frac{\alpha}{\xi^{\alpha+1}}, \quad \forall x \geq 0.$$

By Lemma 1, $\varphi(x)$ is $\frac{\alpha}{\xi^{\alpha+1}}$ -Lipschitz continuous. Then by Definition 1,

$$|\varphi(x) - \varphi(y)| \leq \frac{\alpha}{\xi^{\alpha+1}} |x - y|, \quad \forall x \geq 0, y \geq 0. \quad (7)$$

By Eq. (6),

$$\nabla U(\mathbf{x}) = \nabla \left[\sum_{s \in S} U_{\alpha, \xi}(x_s) \right] = ((x_s + \xi)^{-\alpha})_{s \in S}. \quad (8)$$

Then for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^d$, we have:

$$\begin{aligned} \|\nabla U(\mathbf{x}) - \nabla U(\mathbf{y})\| &= \sqrt{\sum_{s \in S} [(x_s + \xi)^{-\alpha} - (y_s + \xi)^{-\alpha}]^2} \quad (\text{by Eq. (8)}) \\ &\leq \sqrt{\sum_{s \in S} \left[\frac{\alpha}{\xi^{\alpha+1}} |x_s - y_s| \right]^2} \quad (\text{by Eq. (7)}) \\ &= \frac{\alpha}{\xi^{\alpha+1}} \|\mathbf{x} - \mathbf{y}\|. \end{aligned}$$

By Definition 2, $U(\mathbf{x}) = \sum_{s \in S} U_{\alpha, \xi}(x_s)$ is $\frac{\alpha}{\xi^{\alpha+1}}$ -smooth. $\square$

Remark 1: A special case is when $\alpha = 0$. In this case, the function $U(\mathbf{x}) = \sum_{s \in S} U_0(x_s) = \sum_{s \in S} x_s$ is already 0-smooth, so there is no need to use $U_{0, \xi}(x_s)$. In addition, $\sum_{s \in S} U_{0, \xi}(x_s)$ is also 0-smooth for any $\xi \in \mathbb{R}$. Thus, Lemma 4 still holds for $\alpha = 0$. The proof is omitted due to the space limitation.

C. Smooth Penalty Function

The exact penalty function $P_{e,+}$ is not continuously differential and not smooth. Neither does the corresponding network penalty function. Several works have explored approximating $P_{e,+}$ with functions with more tractable mathematical properties, see [35], [36], and [37] for example. Piece-wise functions are commonly used, which are difficult to handle. And although the functions are continuously differential, their smoothness is not proved.

We observe that the exact penalty function is actually the well-known *Rectified Linear Unit (ReLU)* activation function widely used in deep learning. Inspired by this, we propose to use the *Softplus* function defined as

$$f(x) = \ln(1 + \exp(x)), \quad \forall x \in \mathbb{R},$$

to construct the penalty function. Fig. 1 shows the graphs of ReLU function and Softplus function. The two functions behave similar except near the origin, where the Softplus function is differential and seems to be smooth. Then, for $e \in E$ we define the soft penalty function $P_{e,S}$ as:

$$P_{e,S}(\mathbf{A}_e \mathbf{x}) = \ln(1 + \exp(\mathbf{A}_e \mathbf{x} - c_e)).$$

We then prove that its corresponding network penalty function is smooth.

Lemma 5: The function $P(\mathbf{x}) = \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x})$ is $(\frac{1}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2)$ -smooth on $\mathbb{R}_+^d$.

Proof: For all $e \in E$, we define function $f_e : \mathbb{R}_+ \rightarrow \mathbb{R}_+^d$ as

$$f_e(x) = x \mathbf{A}_e,$$

which is $\|\mathbf{A}_e\|$ -Lipschitz on $\mathbb{R}_+$ by Definition 1, because for all $x, y \in \mathbb{R}_+$, we have

$$\|f_e(x) - f_e(y)\| = \|(x - y) \mathbf{A}_e\| = \|\mathbf{A}_e\| |x - y|.$$

We define function $g : \mathbb{R} \rightarrow \mathbb{R}^+$ as

$$g(x) = f'(x) = \frac{1}{1 + e^{-x}},$$

which is $\frac{1}{4}$ -Lipschitz by Lemma 1, because its derivative satisfies

$$|g'(x)| = \left| \frac{e^{-x}}{(1 + e^{-x})^2} \right| \leq \frac{1}{4}, \quad \forall x \in \mathbb{R}.$$

For all $e \in E$, we define function $h_e : \mathbb{R}_+^d \rightarrow \mathbb{R}$ as

$$h_e(\mathbf{x}) = \mathbf{A}_e \mathbf{x} - c_e,$$

which is $\|\mathbf{A}_e\|$ -Lipschitz on $\mathbb{R}_+^d$ by Definition 1, because for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^d$, we have

$$\|h_e(\mathbf{x}) - h_e(\mathbf{y})\| = \|\mathbf{A}_e \mathbf{x} - \mathbf{A}_e \mathbf{y}\| \leq \|\mathbf{A}_e\| \|\mathbf{x} - \mathbf{y}\|.$$

For all $e \in E$, we define function $\tilde{P}_{e,S}(\mathbf{x}) = P_{e,S}(\mathbf{A}_e \mathbf{x})$, whose gradient is

$$\nabla \tilde{P}_{e,S}(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{A}_e \mathbf{x} + c_e)} \mathbf{A}_e = f_e \circ g \circ h_e(\mathbf{x}).$$

By Lemma 2, $\nabla \tilde{P}_{e,S}(\mathbf{x})$ is $(\frac{1}{4} \|\mathbf{A}_e\|^2)$ -Lipschitz on $\mathbb{R}_+^d$, i.e., $\tilde{P}_{e,S}(\mathbf{x})$ is $(\frac{1}{4} \|\mathbf{A}_e\|^2)$ -smooth on $\mathbb{R}_+^d$. Then by Lemma 3, the function $P(\mathbf{x}) = \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x})$ is $(\frac{1}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2)$ -smooth on $\mathbb{R}_+^d$. $\square$

D. NUM With Smooth Objective Function

With the smooth utility function and smooth penalty function, the NUM problem is now defined as:

$$\begin{aligned} \min V_S(\mathbf{x}) &= - \sum_{s \in S} U_{\alpha, \xi}(x_s) + \mu \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x}) \\ \text{s.t., } x_s &\geq 0, \forall s \in S \end{aligned} \quad (\text{NUM-S})$$

Theorem 1: The function V_S is β_V -smooth on $\mathbb{R}_+^d$, where $\beta_V = \frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 = O(d)$.

Proof: We define $U(\mathbf{x}) = \sum_{s \in S} U_{\alpha, \xi}(x_s)$ and $P(\mathbf{x}) = \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x})$, then

$$\nabla V_S(\mathbf{x}) = -\nabla U(\mathbf{x}) + \mu \nabla P(\mathbf{x}).$$

Then for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^d$, we have

$$\begin{aligned} \|\nabla V_S(\mathbf{x}) - \nabla V_S(\mathbf{y})\| &= \|\nabla U(\mathbf{y}) - \nabla U(\mathbf{x}) + \mu \nabla P(\mathbf{x}) - \mu \nabla P(\mathbf{y})\| \\ &\leq \|\nabla U(\mathbf{y}) - \nabla U(\mathbf{x})\| + \mu \|\nabla P(\mathbf{x}) - \nabla P(\mathbf{y})\| \\ &\leq \left(\frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 \right) \|\mathbf{x} - \mathbf{y}\| \quad (\text{by Lemma 4, 5}) \end{aligned}$$

By Definition 2, the function V_S is $\left(\frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 \right)$ -smooth on $\mathbb{R}_+^d$. Besides, by definition of the routing matrix $\mathbf{A}$ (Eq. (1)), we have:

$$\beta_V = \frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 \leq \frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} |E| d. \quad (9)$$

By Assumption 1 and Assumption 2, $|E| \ll d$. Thus, by Eq. (9), $\beta_V = O(d)$. $\square$

One benefit of (NUM-S) is that Assumption 4 holds for its objective function V_S . For all $\mathbf{x} \in \mathbb{R}_+^d, s \in S$, $\left| \frac{\partial V_S(\mathbf{x})}{\partial x_s} \right| \leq M = \max \{ \xi^{-\alpha}, \mu \sum_{e \in E} \mathbf{A}_{es} \mid s \in S \}$. Thus, PGD and Exp-NUM can actually be used to solve (NUM-S) and the convergence analyses become solid. Furthermore, because the objective function is now smooth, the convergence guarantee of PGD can be improved as follows:

Proposition 3: [Convergence rate of PGD for (NUM-S) [21]] If Algorithm 1 is run for T iterations with an initial point $\mathbf{x}^{(0)}$, step size $\gamma = \frac{1}{\beta_V}$, then the solution $\mathbf{x}^{(T)}$ satisfies:

$$V_S(\mathbf{x}^{(T)}) - V_S^* \leq \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{T}. \quad (10)$$

By Theorem 1, $\beta_V = O(d)$. Thus, the error is $\varepsilon = O(d/t)$ at step t , or equivalently, PGD achieves an ε -optimal solution with $\Omega(d/\varepsilon)$ iterations.

For (NUM-S), the convergence rate of PGD is now proportional to $1/t$ instead of $1/\sqrt{t}$, but the iteration complexity is still proportional to d . Note that now the final solution is not the averaged point, but the point at the last iteration. However, Exp-NUM is only applicable to the minimization of Lipschitz functions, so its convergence rate remains $O(\sqrt{\ln d}/\sqrt{t})$ for (NUM-S). The convergence rate of PGD is faster than Exp-NUM with respect to d for (NUM-S), and it can be proven that the error upper bound of PGD is lower than that of Exp-NUM after enough number of iterations.

Proposition 4: The error upper bound of PGD is lower than the error upper bound of Exp-NUM when the number of iterations is larger than $\frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}$.

Proof: By Eq. (4) and Eq. (10), for PGD to have a lower error upper bound, there should be:

$$\begin{aligned} \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{T} &< \frac{M \sqrt{2CD(\mathbf{x}^*, \mathbf{x}^{(0)})}}{\sqrt{T}}, \\ \implies \frac{T}{\sqrt{T}} &> \frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}, \\ \implies T &> \frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}. \end{aligned}$$

$\square$

VI. ACCELERATED GRADIENT METHOD FOR NUM WITH SMOOTH OBJECTIVE FUNCTION

In this section, we present the accelerated gradient method (AGM), which is a tailor-made algorithm for smooth objective function.

A. Vanilla Accelerated Gradient Method

We have shown that the proposed (NUM-S) problem benefits the use of PGD and Exp-NUM. Furthermore, it also enables the use of the *accelerated gradient method* (AGM) proposed by Nesterov [19], which achieves a superior $O(1/t^2)$ convergence rate for smooth objective functions.

Algorithm 3 Accelerated Gradient Method (AGM)

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T
Output: final solution $\mathbf{x}^{(T)}$

```

1:  $\mathbf{y}^{(0)} \leftarrow \mathbf{x}^{(0)}, a_0 \leftarrow 1$ 
2: for  $t \leftarrow 0$  to  $T - 1$  do
3:    $\mathbf{x}^{(t+1)} \leftarrow [\mathbf{y}^{(t)} - \gamma \nabla V_S(\mathbf{y}^{(t)})]_+$ 
4:    $a_{t+1} \leftarrow (1 + \sqrt{4a_t^2 + 1})/2$ 
5:    $\mathbf{y}^{(t+1)} \leftarrow \mathbf{x}^{(t+1)} + \frac{a_t - 1}{a_{t+1}}(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)})$ 
6: end for
7: return:  $\mathbf{x}^{(T)}$ 
```

The pseudocode of AGM is shown in Algorithm 3. At each iteration, $\mathbf{x}^{(t+1)}$ is moved towards the opposite gradient direction at the point $\mathbf{y}^{(t)}$ instead of $\mathbf{x}^{(t)}$. Then the point $\mathbf{y}^{(t+1)}$ is updated based on the obtained point $\mathbf{x}^{(t+1)}$ at this iteration and $\mathbf{x}^{(t)}$ at previous iteration, balanced by the parameters a_t and a_{t+1} . AGM can be viewed as a momentum method [38], because it takes steps based on information at not only the current iteration but also previous iterations. It has the following convergence guarantee:

Theorem 2 (Convergence rate of AGM [19]): If Algorithm 3 is run for T iterations with an initial point $\mathbf{x}^{(0)}$, step size $\gamma = \frac{1}{\beta_V}$, then the final solution $\mathbf{x}^{(T)}$ satisfies:

$$V_S(\mathbf{x}^{(T)}) - V_S^* \leq \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{(T+1)^2}. \quad (11)$$

By Theorem 1, $\beta_V = O(d)$. Thus, the error is $\varepsilon = O(d/t^2)$ at step t , or equivalently, AGM achieves an ε -optimal solution with $\Omega(\sqrt{d}/\sqrt{\varepsilon})$ iterations.

Like PGD and Exp-NUM, AGM is also a first-order method with low per-iteration cost. In addition, AGM has two main advantages over PGD and Exp-NUM:

- 1) AGM has a fast convergence rate with respect to t , which is proportional to $1/t^2$. Compared with PGD (convergence rate proportional to $1/t$) and Exp-NUM (convergence rate proportional to $1/\sqrt{t}$), AGM is much more efficient, converging to an accurate solution within a lot less iterations.
- 2) The iteration complexity of AGM is proportional to $\sqrt{d}$. Although AGM is not as scalable as Exp-NUM with respect to the number of flows, it can still maintain good performance as the number of flows grows.

Remark 2: The convergence guarantees of PGD, Exp-NUM and AGM hold under different assumptions: Assumption 4 or 3 for PGD (Proposition 1 or 3), Assumption 4-5 for Exp-NUM (Proposition 2), and Assumption 1-3 for AGM (Theorem 1, 2). When Assumption 1-5 are all true, Proposition 2, 3 and Theorem 2 all hold, and it can be proven that the error upper bound of AGM is always lower than that of PGD at every iteration step. Moreover, after a sufficient number of iterations, the error upper bound of AGM also becomes lower than that of Exp-NUM.

Theorem 3: The error upper bound of AGM is lower than that of PGD for any number of iterations T .

Proof: By Eq. (10) and Eq. (11), for AGM to have a lower error upper bound, there should be:

$$\begin{aligned} \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{(T+1)^2} &< \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{T}, \\ \Rightarrow \frac{(T+1)^2}{T} &> 1. \end{aligned} \quad (12)$$

Obviously, Eq. (12) holds for any $T \geq 1$. $\square$

Theorem 4: The error upper bound of AGM is lower than that of Exp-NUM when the number of iterations is larger than $\left(\frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}\right)^{\frac{1}{3}} - 1$.

Proof: By Eq. (4) and Eq. (11), for AGM to have a lower error upper bound, there should be:

$$\begin{aligned} \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{(T+1)^2} &< \frac{M \sqrt{2CD(\mathbf{x}^*, \mathbf{x}^{(0)})}}{\sqrt{T}}, \\ \Rightarrow \frac{(T+1)^4}{T} &> \frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}. \end{aligned} \quad (13)$$

As $\frac{(T+1)^4}{T} > \frac{(T+1)^4}{T+1}$, for Eq. (13) to hold, it is sufficient that the following inequality holds:

$$\frac{(T+1)^4}{T+1} > \frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}. \quad (14)$$

By Eq. (14), $T > \left(\frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}\right)^{\frac{1}{3}} - 1$. $\blacksquare$

Note that the comparison in Proposition 4, Theorem 3 and Theorem 4 is based on the theoretical error upper bounds of the algorithms. The actual performance of PGD, Exp-NUM and AGM may vary depending on the specific problem instance.

B. Accelerated Gradient Method With Restart

Although Theorem 2 guarantees that AGM converges fast, it does not imply that the obtained objective value sequence is monotone. When running the AGM algorithm, it is common to observe bumps in the objective value trajectory, which empirically slows down the convergence. To deal with this problem, we leverage the restart heuristic proposed by O'donoghue et al [39]. They propose to "restart" the algorithm, i.e., taking the current point as a new initial point, when certain restart condition is satisfied. In this paper, we consider two restart conditions:

- 1) function restart: $V_S(\mathbf{x}^{(k+1)}) > V_S(\mathbf{x}^{(k)})$;
- 2) gradient restart: $[\nabla V_S(\mathbf{y}^{(k)})]^\top (\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) > 0$.

The pseudocode of AGM with restart is shown in Algorithm 4. At each iteration, the algorithm checks whether the restart condition holds. If so, $\mathbf{y}^{(t+1)}$ is set to $\mathbf{x}^{(t+1)}$ and the parameter a_{t+1} is reset to 1, so that the algorithm is restarted by the current point. Although the restart method boosts convergence empirically, there is no theoretical guarantee on the convergence rate.

VII. GENERALIZED NUM PROBLEM

In previous sections, we focus on the NUM problem in a relatively simple network scenario: each non-prioritized flow is routed along a single path, the network utility is defined as the sum of individual flow utilities, and the Softplus function is used as the link penalty function. However, practical network

Algorithm 4 Accelerated Gradient Method With Restart

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T
Output: final solution $\mathbf{x}^{(T)}$

```

1:  $\mathbf{y}^{(0)} \leftarrow \mathbf{x}^{(0)}, a_0 \leftarrow 1$ 
2: for  $t \leftarrow 0$  to  $T - 1$  do
3:    $\mathbf{x}^{(t+1)} \leftarrow [\mathbf{y}^{(t)} - \gamma \nabla V_S(\mathbf{y}^{(t)})]_+$ 
4:    $a_{t+1} \leftarrow (1 + \sqrt{4a_t^2 + 1})/2$ 
5:    $\mathbf{y}^{(t+1)} \leftarrow \mathbf{x}^{(t+1)} + \frac{a_t - 1}{a_{t+1}}(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)})$ 
6:   if Restart Condition then
7:      $\mathbf{y}^{(t+1)} \leftarrow \mathbf{x}^{(t+1)}, a_{t+1} \leftarrow 0$ 
8:   end if
9: end for
10: return:  $\mathbf{x}^{(T)}$ 

```

TABLE III

NOTATIONS FOR THE GENERALIZED NUM PROBLEM

Symbol	Description
$\mathcal{P}_s$	The set of pre-defined paths $\mathcal{P}_s$ for flow $s \in S$.
$x_{s,p}$	The rate of flow s on path $p \in \mathcal{P}_s$.
$\mathbf{x}$	$\mathbf{x} = (x_{\phi(s,p)})_{s \in S, p \in \mathcal{P}_s}$ is the vector of flow rates, and $x_{\phi(s,p)}$ denotes the rate of flow s on path $p \in \mathcal{P}_s$.
$\mathbf{A}$	The $ E \times (\sum_{s \in S} \mathcal{P}_s)$ routing matrix where each element $\mathbf{A}_{e,\phi(s,p)} \in [0, 1]$ represents the fraction of flow on link e when flow s is routed along path p .
$\phi(s, p)$	The hash function $\phi(s, p)$ returns a unique integer number between 1 to $\sum_{s \in S} \mathcal{P}_s $ for each $s \in S, p \in \mathcal{P}_s$.
w_s	w_s is a weight factor representing the relative importance or priority of flow s .
U_s, U	U_s is the flow utility function and U is the network utility function, and $U(\mathbf{x}) = \sum_{s \in S} w_s U_s(\sum_{p \in \mathcal{P}_s} x_{s,p})$.

environments are often more complex. For example, in large-scale data center (DC) networks or inter-DC WANs, traffic originates from diverse applications with varying priorities and performance requirements. In such networks, operators often deploy advanced routing strategies such as Segment Routing (SR) [40] combined with multi-path routing, which enables fine-grained control over traffic paths, leading to improved load balancing, fault tolerance, and efficient resource utilization. In this section, we extend the NUM problem to more complex network scenarios, and demonstrate the applicability and effectiveness of AGM for the generalized problem.

A. Network Scenarios

First, we introduce the network scenarios we consider in this section. Table III summarizes the notations.

1) *Multi-Path Routing*: In previous sections, we assume that each flow is routed along only one path. To improve resource utilization, enhance load balancing and increase fault tolerance, network operators often employ multi-path routing, where each flow can be split on multiple paths.

In the multi-path scenario, a set of paths $\mathcal{P}_s$ is pre-defined for each flow $s \in S$. $x_{s,p}$ denotes the rate of flow s on path $p \in \mathcal{P}_s$, and the total rate of flow s is $\sum_{p \in \mathcal{P}_s} x_{s,p}$. To arrange $x_{s,p}$ as a column vector, we assume the existence of a hash function $\phi(s, p)$ that returns a unique integer number between 1 and $\sum_{s \in S} |\mathcal{P}_s|$ for each $s \in S, p \in \mathcal{P}_s$. Then $\mathbf{x} = (x_{\phi(s,p)})_{s \in S, p \in \mathcal{P}_s}$ represents the column vector of flow rates. Note that $x_{s,p}$ and $x_{\phi(s,p)}$ have identical meanings, and we use them interchangeably in this paper.

For the number of paths used per flow, denoted by $|\mathcal{P}_s|$, we make the following assumption:

Assumption 6: The number of paths used per flow $|\mathcal{P}_s|$ does not exceed a number K , and K is much smaller than the number of flows d , i.e., $\max_{s \in S} |\mathcal{P}_s| \leq K \ll d$.

This assumption aligns with common network practices. To fully utilize network resources, balance network traffic, and reduce management overhead, network operators typically select a small number of paths for each flow based on some beneficial selection method. For example, network operators might select the top- K shortest paths or optimal oblivious paths, where K is usually a small number, such as $K = 3$ or $K = 4$ [41], [42].

2) *Complex Path*: In previous sections, besides assuming one-path routing per flow, we also assume that each path p is a simple path, i.e., the flow will fully pass each link along the path without splitting. As a result, the routing matrix $\mathbf{A}$ is a binary matrix, and each element $\mathbf{A}_{e,s}$ is a binary variable indicating whether flow s passes through link e ($\mathbf{A}_{e,s} = 1$) or not ($\mathbf{A}_{e,s} = 0$). This assumption is appropriate for certain network scenarios, such as Multi-Protocol Label Switching (MPLS) [43] or SDN networks with OpenFlow [44], where consecutive links form simple routing tunnels.

However, in other network scenarios, the assumption of simple paths may not hold. Complex routing mechanisms often lead to complex routing paths, where traffic flow splits across multiple links. For instance, in networks using the Open Shortest Path First (OSPF) protocol, traffic can be split across multiple links on the shortest paths between the source and destination nodes using Equal-Cost Multi-Path (ECMP) routing. Similarly, in Segment Routing (SR) networks, a SR-tunnel consists of a sequence of segments, i.e. middlepoints. Between two middlepoints, traffic is steered along the OSPF shortest paths in an ECMP manner.

In multi-path routing, the flow s is routed along multiple paths $p \in \mathcal{P}_s$ with controllable sending rate $x_{s,p}$. In contrast, along a complex path p , the flow splits across multiple links, but the fraction of flow on each link is not controlled by the network operator; instead, it is determined by the routing mechanism (e.g., ECMP) itself. In such scenarios, the routing matrix $\mathbf{A}$ is no longer binary. Following the multi-path routing scenario, the routing matrix $\mathbf{A}$ has dimensions $|E| \times (\sum_{s \in S} |\mathcal{P}_s|)$. Each element $\mathbf{A}_{e,\phi(s,p)}$ is relaxed to $[0, 1]$, representing the fraction of flow on link e when flow s is routed along path p . $\mathbf{A}_e$ denotes the e -th row of matrix $\mathbf{A}$. With the flow rate vector defined as $\mathbf{x} = (x_{\phi(s,p)})_{s \in S, p \in \mathcal{P}_s}$, $\mathbf{A}_e \mathbf{x}$ is still the amount of flow routing on link e .

3) *Weighted Network Utility*: In Section III-A, the *network utility function* $U(\mathbf{x})$ is defined as the sum of *flow utility function* U_s for each flow s . However, different users or applications may have varying levels of importance or priority. For instance, premium users might pay more for better service quality, or critical applications might require higher priority. In such cases, a *weight factor* w_s can be assigned to each flow utility function to represent the relative importance or priority of flow s [10]. Consequently, the network utility function becomes the weighted sum of flow utility functions. Continuing with the multi-path routing scenario, the network utility function is defined as

$$U(\mathbf{x}) = \sum_{s \in S} w_s U_s \left(\sum_{p \in \mathcal{P}_s} x_{s,p} \right). \quad (15)$$

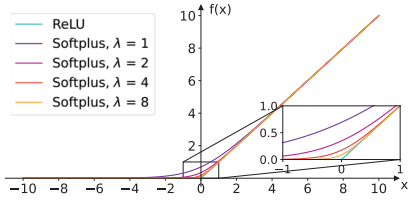


Fig. 2. Graphs of ReLU function and generalized Softplus function with different λ .

4) *Generalized Softplus Penalty Function*: In Section V-C, the Softplus function is used as the smooth penalty function. One concern about the Softplus function is that the penalty value is although small, but still greater than zero when $\mathbf{A}_e \mathbf{x} \leq c_e$. Such unwanted penalty can be mitigated by using the *generalized Softplus function* defined as

$$f(x) = \frac{1}{\lambda} \ln[1 + \exp(\lambda x)], \quad \forall x \in \mathbb{R},$$

where λ is a sharpness parameter controlling the similarity of the generalized Softplus function to the ReLU function. Larger λ allows the penalty function to better approximate the ReLU function.

Fig. 2 shows the graphs of ReLU function and generalized Softplus function with different λ . When $\lambda = 1$, the generalized Softplus function reduces to the standard Softplus function. As λ increases, the generalized Softplus function approximates the ReLU function more closely: $f(x)$ is very close to 0 when $x \leq 0$, and it becomes almost linear for positive x .

When using the generalized Softplus function, the corresponding link penalty function is defined as

$$P_{e,S,\lambda}(\mathbf{A}_e \mathbf{x}) = \frac{1}{\lambda} \ln\{1 + \exp[\lambda(\mathbf{A}_e \mathbf{x} - c_e)]\}.$$

B. Generalized Problem Formulation

Considering the extended network scenarios, the generalized network utility maximization problem is formulated as:

$$\begin{aligned} \min \quad & \tilde{V}_S(\mathbf{x}) = -\sum_{s \in S} w_s U_{\alpha,\xi} \left(\sum_{p \in \mathcal{P}_s} x_{s,p} \right) + \mu \sum_{e \in E} P_{e,S,\lambda}(\mathbf{A}_e \mathbf{x}) \\ \text{s.t.,} \quad & x_{s,p} \geq 0, \quad \forall s \in S, p \in \mathcal{P}_s \quad (\text{NUM-Generalized}) \end{aligned}$$

In order for the applicability of AGM to the generalized NUM problem, it is essential that the objective function $\tilde{V}_S(\mathbf{x})$ is a smooth function. We first prove that the weighted network utility function and the generalized Softplus function is smooth separately (Lemma 6 and Lemma 7), then prove the smoothness of $\tilde{V}_S(\mathbf{x})$ (Theorem 5).

Lemma 6: The function $U(\mathbf{x}) = \sum_{s \in S} w_s U_{\alpha,\xi} \left(\sum_{p \in \mathcal{P}_s} x_{s,p} \right)$ is $\left(\frac{\alpha}{\xi^{\alpha+1}} \cdot \max_{s \in S} (|\mathcal{P}_s| w_s) \right)$ -smooth on $\mathbb{R}_+^d$.

Proof: As in the proof of Lemma 4, we define function $\varphi: \mathbb{R}_+ \rightarrow \mathbb{R}_+$ as:

$$\varphi(x) = (x + \xi)^{-\alpha},$$

where $\alpha \geq 0, \xi > 0$. $\varphi(x)$ is $\frac{\alpha}{\xi^{\alpha+1}}$ -Lipschitz continuous and satisfies Eq. (7).

By Eq. (6) and Eq. (15),

$$\begin{aligned} \nabla U(\mathbf{x}) &= \nabla \left[\sum_{s \in S} w_s U_{\alpha,\xi} \left(\sum_{p \in \mathcal{P}_s} x_{\phi(s,p)} \right) \right] \\ &= \left(w_s \left(\sum_{\tilde{p} \in \mathcal{P}_s} x_{\phi(s,\tilde{p})} + \xi \right)^{-\alpha} \right)_{s \in S, p \in \mathcal{P}_s} \end{aligned}$$

Then for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^d$, we have:

$$\begin{aligned} & \|\nabla U(\mathbf{x}) - \nabla U(\mathbf{y})\| \\ &= \sqrt{\sum_{s \in S} \sum_{p \in \mathcal{P}_s} \left[w_s \left(\sum_{\tilde{p} \in \mathcal{P}_s} x_{\phi(s,\tilde{p})} + \xi \right)^{-\alpha} - w_s \left(\sum_{\tilde{p} \in \mathcal{P}_s} y_{\phi(s,\tilde{p})} + \xi \right)^{-\alpha} \right]^2} \\ &= \sqrt{\sum_{s \in S} |\mathcal{P}_s| w_s^2 \left[\left(\sum_{\tilde{p} \in \mathcal{P}_s} x_{\phi(s,\tilde{p})} + \xi \right)^{-\alpha} - \left(\sum_{\tilde{p} \in \mathcal{P}_s} y_{\phi(s,\tilde{p})} + \xi \right)^{-\alpha} \right]^2} \\ &\quad (\text{by the Lipschitz continuity of } \varphi, \text{ i.e., Eq. (7)}) \\ &\leq \sqrt{\sum_{s \in S} |\mathcal{P}_s| w_s^2 \left[\frac{\alpha}{\xi^{\alpha+1}} \left(\sum_{\tilde{p} \in \mathcal{P}_s} x_{\phi(s,\tilde{p})} - \sum_{\tilde{p} \in \mathcal{P}_s} y_{\phi(s,\tilde{p})} \right) \right]^2} \\ &\quad (\text{by Cauchy-Schwarz Inequality}) \\ &\leq \frac{\alpha}{\xi^{\alpha+1}} \sqrt{\sum_{s \in S} \left[|\mathcal{P}_s|^2 \cdot w_s^2 \cdot \sum_{\tilde{p} \in \mathcal{P}_s} (x_{\phi(s,\tilde{p})} - y_{\phi(s,\tilde{p})})^2 \right]} \\ &\leq \frac{\alpha}{\xi^{\alpha+1}} \sqrt{\max_{s \in S} (|\mathcal{P}_s|^2 w_s^2) \cdot \sum_{s \in S} \sum_{\tilde{p} \in \mathcal{P}_s} (x_{\phi(s,\tilde{p})} - y_{\phi(s,\tilde{p})})^2} \\ &= \frac{\alpha}{\xi^{\alpha+1}} \cdot \max_{s \in S} (|\mathcal{P}_s| w_s) \cdot \|\mathbf{x} - \mathbf{y}\| \end{aligned}$$

By Definition 2, $U(\mathbf{x}) = \sum_{s \in S} w_s U_{\alpha,\xi} \left(\sum_{p \in \mathcal{P}_s} x_{s,p} \right)$ is $\left(\frac{\alpha}{\xi^{\alpha+1}} \cdot \max_{s \in S} (|\mathcal{P}_s| w_s) \right)$ -smooth. ■

Lemma 7: The function $P(\mathbf{x}) = \sum_{e \in E} P_{e,S,\lambda}(\mathbf{A}_e \mathbf{x})$ is $\left(\frac{\lambda}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 \right)$ -smooth on $\mathbb{R}_+^d$.

The proof of Lemma 7 is very similar to the proof of Lemma 5. The main difference is defining

$$g(x) = f'(x) = \frac{1}{1 + e^{-\lambda x}},$$

and $g(x)$ is $\frac{\lambda}{4}$ -Lipschitz by Lemma 1, because its derivative satisfies

$$|g'(x)| = \left| -\lambda \frac{e^{-\lambda x}}{(1 + e^{-\lambda x})^2} \right| \leq \frac{\lambda}{4}, \quad \forall x \in \mathbb{R}.$$

We omit the detailed proof to avoid redundancy.

Theorem 5: The function $\tilde{V}_S(\mathbf{x})$ is $\beta_{\tilde{V}}$ -smooth on $\mathbb{R}_+^d$, where $\beta_{\tilde{V}} = \left(\frac{\alpha}{\xi^{\alpha+1}} \cdot \max_{s \in S} (|\mathcal{P}_s| w_s) \right) + \frac{\mu\lambda}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 = O(d)$.

The proof of Theorem 5 is very similar to the proof of Theorem 1. So we omit the detailed proof to avoid redundancy. The main difference lies in how to prove that $\beta_{\tilde{V}}$ is $O(d)$. With the definition of the routing matrix $\mathbf{A}$ in § VII-A2, $\mathbf{A}_e$ is a $1 \times \sum_{s \in S} |\mathcal{P}_s|$ vector, and each element in $\mathbf{A}_e$ satisfies $0 \leq \mathbf{A}_{e,\phi(s,p)} \leq 1$. As defined in § VII-A3, $0 < w_s < 1$. By Assumption 1 and Assumption 2, $|E| \ll d$. By Assumption 6, $\max_{s \in S} |\mathcal{P}_s| = K$ and $K \ll d$. Thus,

$$\begin{aligned} \beta_{\tilde{V}} &= \left(\frac{\alpha}{\xi^{\alpha+1}} \cdot \max_{s \in S} (|\mathcal{P}_s| w_s) \right) + \frac{\mu\lambda}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 \\ &\leq \frac{\alpha}{\xi^{\alpha+1}} K + \frac{\mu\lambda}{4} |E| K d = O(d). \end{aligned}$$

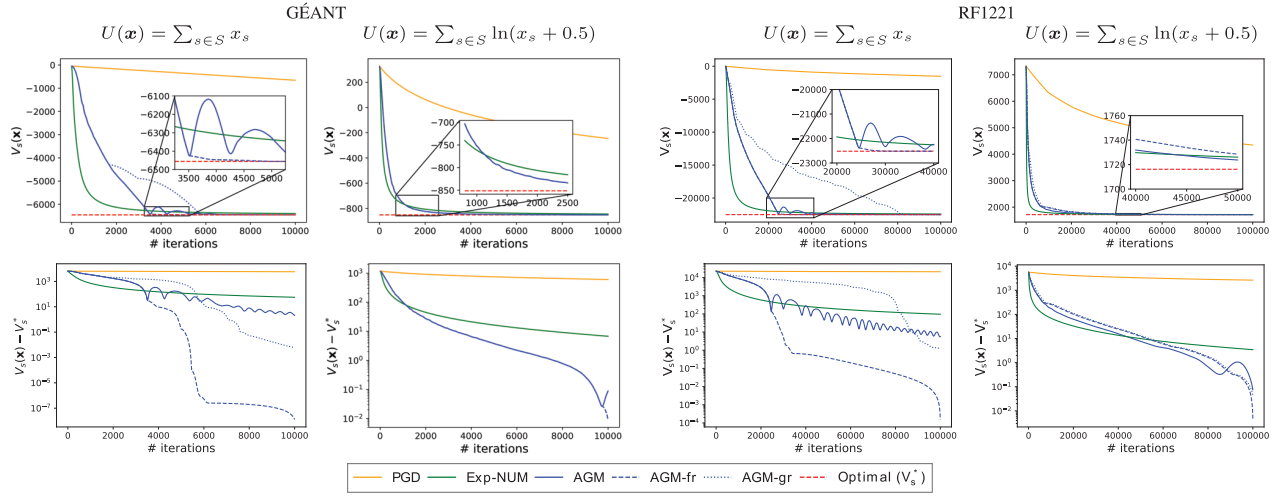


Fig. 3. The objective value and error at each iteration in GÉANT and RF1221.

C. Convergence Rate of the Algorithms

The objective function of the generalized NUM problem $\tilde{V}_S$ is smooth, therefore AGM is applicable for solving this problem. The convergence rate of solving the problem using AGM remains consistent with Theorem 2, except that V_S in Eq. (11) should be replaced by $\tilde{V}_S$, and β_V should be replaced by $\beta_{\tilde{V}}$. As stated in Theorem 5, $\beta_{\tilde{V}}$ is $O(d)$, so the convergence rate of AGM for the generalized NUM problem is still $\varepsilon = O(d/t^2)$, i.e., $t = \Omega(\sqrt{d}/\sqrt{\varepsilon})$.

For PGD, since $\tilde{V}_S$ is smooth, its convergence rate for the generalized NUM problem remains consistent with Proposition 3, with V_S being replaced by $\tilde{V}_S$ and β_V being replaced by $\beta_{\tilde{V}}$.

Regarding Exp-NUM, $\tilde{V}_S$ satisfies Assumption 1, specifically, $\left| \frac{\partial V_p(\mathbf{x})}{\partial x_s} \right| \leq \tilde{M} = \max \{w_s \xi^{-\alpha}, \mu \sum_{e \in E} \mathbf{A}_{e,\phi(s,p)} \forall s \in S, p \in \mathcal{P}_s\}$. Therefore, the convergence rate of Exp-NUM for solving the generalized NUM problem adheres to Proposition 2, where V_p should be replaced by $\tilde{V}_S$ and M should be replaced by $\tilde{M}$.

Compared with the simple NUM problem (NUM-S) defined in Section V-D, the convergence rates of PGD, Exp-NUM, and AGM remain largely unchanged for the generalized NUM problem (NUM-Generalized). Consequently, AGM retains its key advantage for the generalized NUM problem: it achieves a fast convergence rate with respect to t which is proportional to $1/t^2$, and it is scalable with respect to the number of flows d since its iteration complexity is proportional to $\sqrt{d}$. Furthermore, as established in Theorem 3 and Theorem 4, the error upper bound of AGM is lower than that of PGD for any number of iterations, and becomes lower than that of Exp-NUM when the number of iterations exceeds

$$\left(\frac{2\beta_{\tilde{V}} \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{\tilde{M}^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})} \right)^{\frac{1}{3}} - 1.$$

VIII. EVALUATION

In this section, we evaluate the smooth penalty-based NUM formulation NUM-S and the proposed algorithm.

A. Setup

First we introduce the setup for evaluating the performance of AGM for the simple smooth penalty-based NUM problem NUM-S.

1) *Topology*: We use two topologies GÉANT (23 nodes, 74 links) [45] and RF1221 (104 nodes, 302 links) [46]. The simple shortest path without ECMP is used as the one predetermined path for each flow, so elements in the routing matrix are integers. To better display the result, the link capacities are scaled down so that $c_e \leq 100, \forall e \in E$. Because the flow rates x_s are decision variables in the NUM problem, the traffic matrix is not needed in the experiments.

2) *Optimization Objective*: We use two different utility functions. One is $\alpha = 0, \xi = 0$, so that the network utility function is $U(\mathbf{x}) = \sum_{s \in S} x_s$, corresponding to throughput maximization. The other is $\alpha = 1, \xi = 0.5$, so that network utility function is $U(\mathbf{x}) = \sum_{s \in S} \ln(x_s + 0.5)$, corresponding to proportional fairness. The Softplus function is used as the link penalty function, i.e., $P_{e,S}(\mathbf{A}_e \mathbf{x}) = \ln(1 + \exp(\mathbf{A}_e \mathbf{x} - c_e))$. μ is set to 2 empirically.

3) *Algorithms*: We solve the problem (NUM-S) with PGD, Exp-NUM (C is set to $\sum_{e \in E} c_e$), AGM, AGM with function restart (i.e., AGM-fr) and AGM with gradient restart (i.e., AGM-gr). All algorithms are implemented in Python. Computations are performed on a server with 2.40GHz CPU and 512G memory.

B. Convergence Rate

We evaluate the convergence rate of the algorithms for the problem (NUM-S). Assuming one flow between each source-destination pair, and run the algorithms with the aforementioned objective. Fig. 3 shows the objective value (at top) and the error (at bottom in log scale) in GÉANT and RF1221 at the first 10000 and 100000 iterations respectively.

First, experiments show that AGM has the fastest convergence rate and outperforms PGD and Exp-NUM, which agrees with the theoretical results. Given a tolerable error, our method will solve the NUM problem much more quickly, which benefits the network resource allocation. As proved in Theorem 3, PGD converges the slowest in experiment, mainly because its iteration complexity is $\Omega(d/\varepsilon)$, which is proportion to d . When comparing AGM and Exp-NUM, the experiment results is in consistent with Theorem 4. Exp-NUM has $O(\sqrt{\ln d}/\sqrt{t})$ convergence rate, which is nearly independent on the number of flows d , thus it converges fast in the beginning. However, the convergence slows down later and Exp-NUM can only achieve an error of $10 \sim 10^2$ by

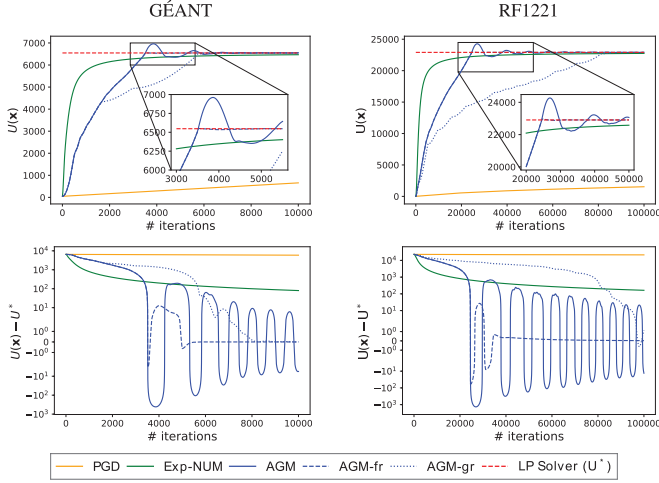


Fig. 4. The network utility value and error at each iteration when $U_s = \sum_{s \in S} x_s$ in GÉANT and RF1221.

the end. In contrast, AGM converges a little slower at first, but it performs better and obtains more accurate solutions as the number of iterations grows, because of its $O(d/t^2)$ convergence rate.

Second, the restart method boosts AGM's convergence and alleviates the bumps. Although AGM converges slightly faster initially, AGM-fr and AGM-gr outperform it as the iterations continue. In particular, AGM-fr achieves 10^{-2} -optimal solution for all objective functions and topologies, and it even achieves 10^{-7} -optimal solution for $U_s = \sum_{s \in S} x_s$ in GÉANT. Besides, there are less bumps in the curves of AGM-fr and AGM-gr.

C. Network Utility Optimization

The objective of (NUM-S) is the smooth penalized function $V_s(x)$, rather than the network utility function $U(x)$. In order to evaluate the performance of the algorithms in terms of network utility maximization, we use $U(x) = \sum_{s \in S} x_s$ as an example so that the problem (NUM) is a linear programming (LP) problem aiming at maximizing throughput and can be solved by commercial LP solvers. We denote its optimal value as U^* . We record the $U(x)$ (at top) and error with respect to U^* (at bottom in symmetric log scale) during iterations when solving (NUM-S). The results are shown in Fig. 4.

First, the rate at which $U(x)$ approaches U^* is consistent with the convergence rate of the algorithms, i.e., Exp-NUM performs well at first but AGM surpasses it later and PGD hardly converges. Hence, the convergence rate reflects the efficiency of the algorithms in solving NUM problem.

Second, AGM causes obvious bumps in $U(x)$. Although it converges faster than Exp-NUM, it tends to overshoot the solution and oscillates around U^* . In contrast, AGM-fr produces stable results while maintaining fast convergence.

Third, AGM-fr achieves close-to-optimal network utility value, which means that the smooth penalty-based formulation and our algorithm can help solve the NUM problem effectively. Table IV presents the $U(x)$ and corresponding error at the last iteration. Still, PGD does not converge, and the error achieved by Exp-NUM is unsatisfactory. AGM overshoots the result while the restart methods perform well. AGM-fr achieves highly accurate $U(x)$ in both topologies.

TABLE IV

THE ACHIEVED NETWORK UTILITY VALUE AND ERROR (IN PARENTHESES) WHEN $U_s = \sum_{s \in S} x_s$ IN GÉANT AND RF1221

Algorithm	GÉANT	RF1221
PGD	657.09 (5890.11)	1533.36 (21374.64)
Exp-NUM	6467.39 (79.81)	22744.75 (163.26)
AGM	6552.78 (-5.58)	22921.28 (-13.28)
AGM-fr	6547.199 (0.001)	22907.97 (0.03)
AGM-gr	6547.19 (0.01)	22906.87 (1.13)
LP Solver	6547.20	22908.00

TABLE V

AVERAGE COMPUTATION TIME PER ITERATION WITH DIFFERENT NUMBER OF FLOWS d WHEN $U_s = \sum_{s \in S} \ln(x_s + 0.5)$ IN GÉANT

Algorithm	$d=100$	$d=1000$	$d=10000$
PGD	$7.2 \times 10^{-5} s$	$1.7 \times 10^{-4} s$	$1.1 \times 10^{-3} s$
Exp-NUM	$9.3 \times 10^{-5} s$	$2.7 \times 10^{-4} s$	$1.3 \times 10^{-3} s$
AGM	$8.0 \times 10^{-5} s$	$2.0 \times 10^{-4} s$	$1.2 \times 10^{-3} s$
AGM-fr	$1.5 \times 10^{-4} s$	$3.4 \times 10^{-4} s$	$1.8 \times 10^{-3} s$
AGM-gr	$8.4 \times 10^{-5} s$	$2.3 \times 10^{-4} s$	$1.3 \times 10^{-3} s$

D. Performance With Different d

We evaluate the impact of the number of flows d on algorithm performance. We take GÉANT with $U_s = \sum_{s \in S} \ln(x_s + 0.5)$ for example, and run the algorithms with 100, 1000, 10000 flows in the network respectively. The results are similar for RF1221 and $U_s = \sum_{s \in S} x_s$.

Fig. 5 shows the error at each iteration. The iteration complexity of AGM is proved to be $\Omega(\sqrt{d}/\epsilon)$, allowing AGM to maintain good performance as d increases. And the experiment results are consistent with the theoretical results. As shown in Fig. 5, different values of d has no obvious effect on the convergence of AGM, it performs consistently well. Besides, although the iteration complexity of Exp-NUM is claimed to be nearly independent on d , our method AGM, AGM-fr and AGM-gr outperforms Exp-NUM with all d settings.

In Table V we record the average computation time per iteration of different algorithms. As d increases, the computation time per iteration increases accordingly. With the same d , the time per iteration of PGD, Exp-NUM, and AGM are similar while Exp-NUM takes a little longer time for the computation of the exponential function, because they are all first-order methods. Thus, the difference in their convergence rates implies their different computation time to solve the problem. AGM-fr takes longer time because of the additional objective value computation when judging restart condition. Given its fast and stable convergence, we think the trade-off is worthwhile.

IX. EVALUATION FOR THE GENERALIZED NUM PROBLEM

In this section, we evaluate the performance of AGM for the generalized NUM problem.

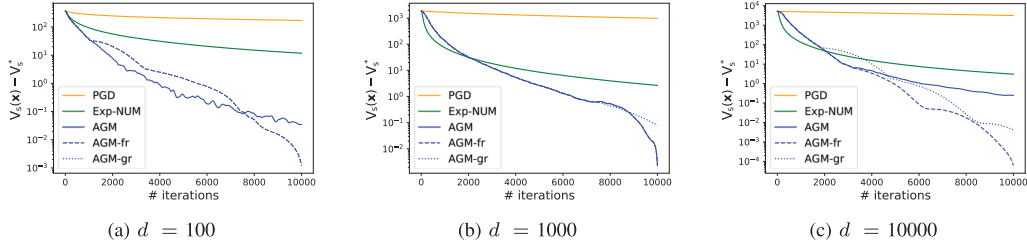


Fig. 5. The error at each iteration with different number of flows d when $U_s = \sum_{s \in S} \ln(x_s + 0.5)$ in GÉANT.

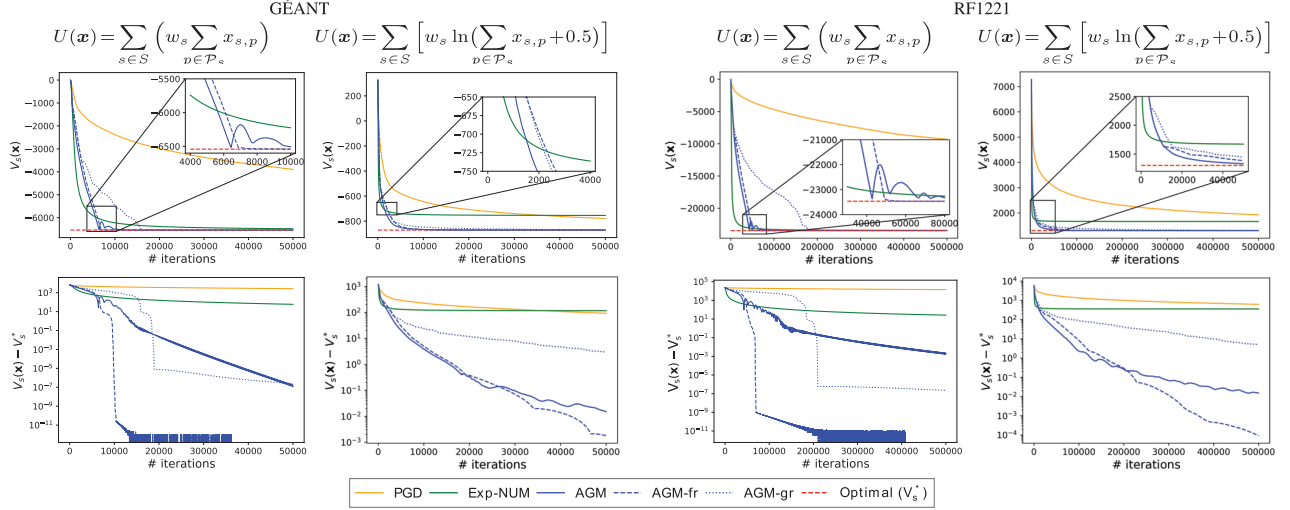


Fig. 6. The objective value and error at each iteration in GÉANT and RF1221 for the generalized NUM problem.

A. Setup

1) *Topology*: We continue to use the two topologies GÉANT (23 nodes, 74 links) and RF1221 (104 nodes, 302 links). The top- K shortest paths are used as the predetermined paths $\mathcal{P}_s$ for each flow s , and we set $K = 3$. For each path, the flow is routed on it in an ECMP manner, so the elements in the routing matrix are real numbers between 0 and 1. To better illustrate the result, the link capacities are still scaled down so that $c_e \leq 100, \forall e \in E$.

2) *Optimization Objective*: We use weighted network utility functions. The weight factor w_s of each flow s is randomly drawn from the uniform distribution over $[1, 5]$. To better display the results, the weight factors are scaled so that $\sum_{s \in S} w_s = d$. We continue to use the two utility function settings $\alpha = 0, \xi = 0$ and $\alpha = 1, \xi = 0.5$. Consequently, the network utility functions used in the evaluation are $U(\mathbf{x}) = \sum_{s \in S} (w_s \sum_{p \in \mathcal{P}_s} x_{s,p})$ and $U(\mathbf{x}) = \sum_{s \in S} [w_s \ln(\sum_{p \in \mathcal{P}_s} x_{s,p} + 0.5)]$. For the link penalty function, we use the generalized Softplus function with $\lambda = 16$, i.e., $P_{e,S,\lambda}(\mathbf{A}_e \mathbf{x}) = \frac{1}{16} \ln\{1 + \exp[16(\mathbf{A}_e \mathbf{x} - c_e)]\}$. μ is empirically set to 2.

3) *Algorithms*: We solve the problem (NUM-Generalized) with PGD, Exp-NUM (C is set to $\sum_{e \in E} c_e$), AGM, AGM with function restart (AGM-fr) and AGM with gradient restart (AGM-gr). All algorithms are implemented in Python. Computations are performed on a server with 2.40GHz CPU and 512G memory.

B. Convergence Rate

We evaluate the convergence rate of the algorithms for the problem (NUM-Generalized). We run the algorithms with the

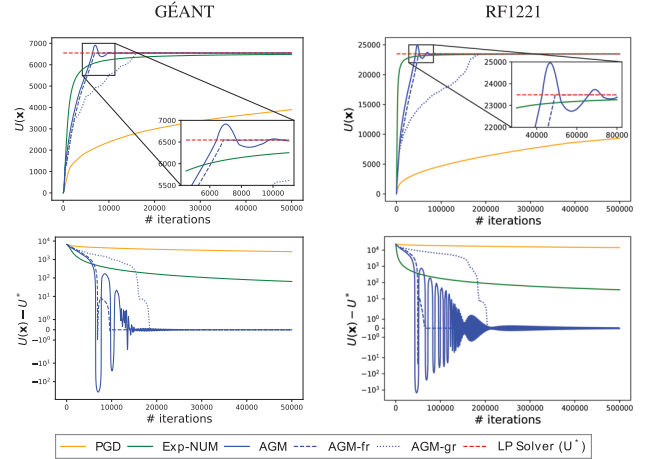


Fig. 7. The network utility value and error at each iteration when $U(\mathbf{x}) = \sum_{s \in S} (w_s \sum_{p \in \mathcal{P}_s} x_{s,p})$ for the generalized NUM problem.

aforementioned evaluation setup, and plot the objective value (at top) and corresponding error (at bottom in log scale) in GÉANT and RF1221 at the first 50000 and 500000 iterations respectively in Fig. 6.

For the generalized NUM problem, AGM continues to demonstrate the fastest convergence rate compared with PGD and Exp-NUM. AGM consistently outperforms PGD throughout the iteration process, and surpasses Exp-NUM after a certain number of iterations, which aligns with the theoretical analysis in Section VII-C. The restart methods still enhance the convergence of AGM. AGM-fr outperforms AGM across both topologies and objectives. In addition, for the

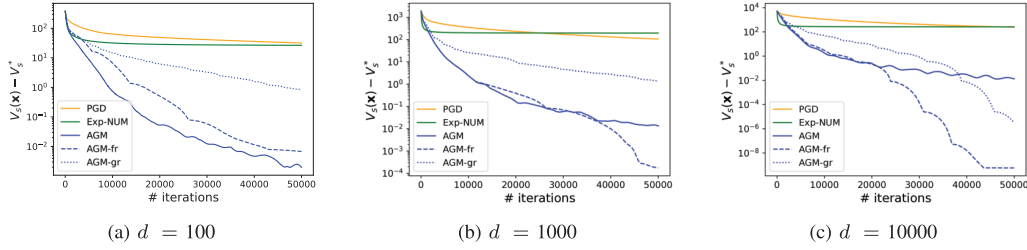


Fig. 8. The error at each iteration with different number of flows d when $U(\mathbf{x}) = \sum_{s \in S} \left[w_s \ln \left(\sum_{p \in \mathcal{P}_s} x_{s,p} + 0.5 \right) \right]$ in GÉANT for the generalized NUM problem.

objective $U(\mathbf{x}) = \sum_{s \in S} \left[w_s \ln \left(\sum_{p \in \mathcal{P}_s} x_{s,p} + 0.5 \right) \right]$, it can be observed from Fig. 6 that PGD converges slow initially, but it converges faster than Exp-NUM as the number of iterations grows, and even achieves better solution in GÉANT, which is in consistent with Proposition 4. The experimental results validate our theoretical analysis, showing that AGM can be applied to the generalized NUM scenario with multiple complex paths and weighted utility functions, and maintains its superior convergence speed.

C. Network Utility Optimization

For the generalized NUM problem, we use $U(\mathbf{x}) = \sum_{s \in S} \left(w_s \sum_{p \in \mathcal{P}_s} x_{s,p} \right)$ as an example to evaluate the performance of the algorithms in terms of network utility maximization. For this LP problem, the optimal network utility value U^* is obtained using a commercial LP solver. Fig. 7 shows the $U(\mathbf{x})$ (at top) and error with respect to U^* (at bottom in symmetric log scale) during iterations when solving (NUM-Generalized).

For the generalized NUM problem, the speed at which each algorithm reaches the optimal network utility value is still consistent with their convergence rate. AGM achieves more accurate $U(\mathbf{x})$ compared to PGD and Exp-NUM. Moreover, the restart methods AGD-fr and AGD-gr help reduce oscillations during iterations, resulting in more stable solutions. The experiment results demonstrate that the smooth penalty-based formulation with generalized Softplus link penalty function, and our algorithm AGM, is effective in solving the NUM problems involving multiple complex routing paths and weighted utility functions.

D. Performance With Different d

To evaluate the impact of the number of flows d for the generalized NUM problem NUM-Generalized, we take GÉANT with $U(\mathbf{x}) = \sum_{s \in S} \left[w_s \ln \left(\sum_{p \in \mathcal{P}_s} x_{s,p} + 0.5 \right) \right]$ as an example, and run the algorithms with 100, 1000, 10000 flows in the network respectively. Fig. 8 shows the error at each iteration step in log scale.

As discussed in Section VII-C, the iteration complexity of AGM remains proportional to $\sqrt{d}$ for the generalized NUM problem, so AGM remains scalable with respect to the number of flows d for the generalized network scenario. As shown in Fig. 8, the convergence rate of AGM shows no significant degradation as the number of flows increases, and AGM-fr even achieves an error less than 10^{-8} when $d = 10000$. As noted in Proposition 4, although Exp-NUM converges fast at the beginning, PGD converges faster than it after a sufficient number of iterations.

X. DISCUSSION

1) Smooth Utility and Penalty Functions:

Our method AGM aims at solving NUM problem with smooth utility functions. Although the most commonly studied α -fair utility function is proven to be smooth, in general, however, utility functions are not necessarily smooth or even not concave, which limits the applicability of our method. Besides, one concern about the Softplus function is that the penalty value is although small, but still greater than zero when $\mathbf{A}_e \mathbf{x} \leq \mathbf{c}_e$. Such unwanted penalty can be mitigated by using the generalized Softplus function with large λ as discussed in Section VII-A.4.

2) Dense Networks:

In this paper, we prove that AGM achieves an iteration complexity of $\Omega(\sqrt{d}/\sqrt{\epsilon})$ under the assumption of sparse network topology, i.e., the number of links $|E|$ is much smaller than the number of flows d . In dense networks, $|E| = \Theta(|V|(|V|-1))$. By Assumption 2, d is at least the same order of magnitude as the number of node pairs, thus $|E| = O(d)$, and the iteration complexity of AGM increases to $\Omega(d/\sqrt{\epsilon})$, indicating reduced scalability with respect to d . To address this, one approach is to limit the average path length of all flows to be at most M and $M \ll d$, which is often realistic in dense networks. Under this assumption, the iteration complexity will revert to $\Omega(\sqrt{d}/\sqrt{\epsilon})$, restoring the scalability of AGM.

3) Large-Scale Networks:

This paper demonstrates that AGM is efficient and scalable, with both theoretical analysis and numerical experiment on topologies with up to 100 nodes. However, in large-scale networks with hundreds of thousands of nodes and flows, the computational resources required by AGM will inevitably increase. First, as the network size and the number of flows grows, both the total number of iterations ($\Omega(\sqrt{d}/\sqrt{\epsilon})$) and the computation time per iteration ($O(|E|d)$) increase, requiring additional computational effort. Second, the memory requirements also increase, mainly due to the storage of the routing matrix and flow information. Nevertheless, AGM mainly relies on basic numerical operations and simple data structure, so the computational resource requirements remain manageable for large-scale networks.

4) Deployment in SDN Networks:

AGM is compatible with existing SDN frameworks, and can be implemented either as a module within the SDN controller, or as an external application communicating with the

controller via the northbound interface. The controller gathers network state information such as link load and flow requests through the southbound interface. AGM then solves the NUM problem and sends back the optimal flow sending rates to the controller for implementation. While deploying AGM in SDN networks simplifies flow rate control and bandwidth resource allocation, it also introduces computational, storage and communication overhead to the controller and network devices.

5) Decentralized Implementation:

AGM is primarily designed as a centralized algorithm, but it can also be implemented in a decentralized manner under certain conditions: each router has the ability to monitor the traffic on its connected link, and there exists a mechanism for each flow source to obtain the sum of link penalties along its routing path [47]. In this scenario, each flow source can independently update its flow sending rate, thus reducing unnecessary global information sharing and enabling parallelism. However, the decentralized implementation introduces new requirements on both the devices and protocols in the network.

XI. CONCLUSION

In this paper, we propose a centralized, efficient and scalable accelerated gradient method (AGM) for the centralized NUM problem. We discover the smoothness of the α -fair utility function, and formulate the NUM problem with smooth objective function, which not only benefits the existing method, but also enables the use of Nesterov's accelerated gradient method. We prove that AGM has a fast $O(d/t^2)$ convergence rate with respect to t , and has $\Omega(\sqrt{d}/\sqrt{\varepsilon})$ iteration complexity which is scalable with respect to the number of flows d . Our smooth penalty-based formulation of the NUM problem and the algorithm AGM are not only applicable to simple network scenarios with single-path routing, but also to complex and practical network scenarios where prioritized flows use multiple complex path routing. Experiment results are in good agreement with the theoretical results and demonstrate the great potential of AGM for resource allocation in large-scale centralized controlled networks. In the future, we will deploy and evaluate AGM in SDN networks with real-world network traffic to further validate its practicality and efficiency.

REFERENCES

- [1] F. Kelly, "Charging and rate control for elastic traffic," *Eur. Trans. Telecommun.*, vol. 8, no. 1, pp. 33–37, 1997.
- [2] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *Proc. SPIE*, vol. 3530, pp. 55–63, Oct. 1998.
- [3] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, Oct. 2000.
- [4] A. Eryilmaz and R. Srikant, "Joint congestion control, routing, and MAC for stability and fairness in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1514–1524, Aug. 2006.
- [5] N. Karakoç, A. Scaglione, A. Nedic, and M. Reisslein, "Multi-layer decomposition of network utility maximization problems," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2077–2091, Oct. 2020.
- [6] L. Gu et al., "Fairness-aware dynamic rate control and flow scheduling for network utility maximization in network service chain," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1059–1071, May 2019.
- [7] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, Apr. 1998.
- [8] S. H. Low and D. E. Lapsley, "Optimization flow control. I. Basic algorithm and convergence," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, pp. 861–874, Dec. 1999.
- [9] R. J. La and V. Anantharam, "Utility-based rate control in the internet for elastic traffic," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 272–286, Apr. 2002.
- [10] S. Shakkottai and R. Srikant, "Network optimization and control," *Found. Trends Netw.*, vol. 2, no. 3, pp. 271–379, Jan. 2007.
- [11] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, Aug. 2006.
- [12] H. Yu and M. J. Neely, "A simple parallel algorithm with an $O(1/t)$ convergence rate for general convex programs," *SIAM J. Optim.*, vol. 27, no. 2, pp. 759–783, Jan. 2017.
- [13] A. Beck, A. Nedic, A. Ozdaglar, and M. Teboulle, "An $O(1/k)$ gradient method for network resource allocation problems," *IEEE Trans. Control Netw. Syst.*, vol. 1, no. 1, pp. 64–73, Mar. 2014.
- [14] Open Networking Foundation, "Software-defined networking: The new norm for networks," USA, White Paper, 2012.
- [15] S. Jain et al., "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.
- [16] C.-Y. Hong et al., "Achieving high utilization with software-driven WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Aug. 2013.
- [17] R. Gupta, L. Vandenbergh, and M. Gerla, "Centralized network utility maximization over aggregate flows," in *Proc. 14th Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, May 2016, pp. 1–8.
- [18] L. Vigneri, G. Paschos, and P. Mertikopoulos, "Large-scale network utility maximization: Countering exponential growth with exponentiated gradients," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 1630–1638.
- [19] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," in *Proc. Dokl. Akad. Nauk Sssr*, vol. 269, 1983, pp. 543–547.
- [20] Y. Nesterov and V. Shikhman, "Dual subgradient method with averaging for optimal resource allocation," *Eur. J. Oper. Res.*, vol. 270, no. 3, pp. 907–916, Nov. 2018.
- [21] S. Bubeck, "Convex optimization: Algorithms and complexity," *Found. Trends Mach. Learn.*, vol. 8, nos. 3–4, pp. 231–357, 2015.
- [22] Z. Allybokus, K. Avrachenkov, J. Leguay, and L. Maggi, "Real-time fair resource allocation in distributed software defined networks," in *Proc. 29th Int. Teletraffic Congr. (ITC)*, vol. 1, Sep. 2017, pp. 19–27.
- [23] M. Ashour, J. Wang, C. Lagoa, N. Aybat, and H. Che, "Non-concave network utility maximization: A distributed optimization approach," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2017, pp. 1–9.
- [24] E. Wei, A. Ozdaglar, and A. Jadbabaie, "A distributed Newton method for network utility maximization," in *Proc. 49th IEEE Conf. Decis. Control (CDC)*, Dec. 2010, pp. 1816–1821.
- [25] M. Zargham, A. Ribeiro, and A. Jadbabaie, "Accelerated backpressure algorithm," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2013, pp. 2269–2275.
- [26] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie, "Accelerated dual descent for network flow optimization," *IEEE Trans. Autom. Control*, vol. 59, no. 4, pp. 905–920, Apr. 2014.
- [27] N. Karakoç, A. Scaglione, M. Reisslein, and R. Wu, "Federated edge network utility maximization for a multi-server system: Algorithm and convergence," *IEEE/ACM Trans. Netw.*, vol. 30, no. 5, pp. 2002–2017, Oct. 2022.
- [28] A. S. Nemirovskij and D. B. Yudin, *Problem Complexity and Method Efficiency in Optimization*. Hoboken, NJ, USA: Wiley, 1983.
- [29] J. Chen, Y. Wang, and T. Lan, "Bringing fairness to actor-critic reinforcement learning for network utility optimization," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Feb. 2021, pp. 1–10.
- [30] H. Hao, W. Ding, and W. Zhang, "Time-continuous computing offloading algorithm with user fairness guarantee," *J. Netw. Comput. Appl.*, vol. 223, Mar. 2024, Art. no. 103826.
- [31] Y. Tian, Z. Wang, X. Yin, X. Shi, J. Yang, and H. Zhang, "Centralized network utility maximization with accelerated gradient method," in *Proc. IEEE 30th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2022, pp. 1–11.
- [32] L. Massoulié and J. Roberts, "Bandwidth sharing: Objectives and algorithms," in *Proc. IEEE INFOCOM Conf. Comput. Commun. 18th Annu. Joint Conf. IEEE Comput. Commun. Societies. Future Now*, Mar. 1999, pp. 1395–1403.
- [33] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.
- [34] W. I. Zangwill, "Non-linear programming via penalty functions," *Manag. Sci.*, vol. 13, no. 5, pp. 344–358, Jan. 1967.

- [35] M. Ç. Pinar and S. A. Zenios, "On smoothing exact penalty functions for convex constrained optimization," *SIAM J. Optim.*, vol. 4, no. 3, pp. 486–511, Aug. 1994.
- [36] X. Xu, Z. Meng, J. Sun, L. Huang, and R. Shen, "A second-order smooth penalty function algorithm for constrained optimization problems," *Comput. Optim. Appl.*, vol. 55, no. 1, pp. 155–172, May 2013.
- [37] S. A. Zenios, M. Ç. Pinar, and R. S. Dembo, "A smooth penalty function algorithm for network-structured problems," *Eur. J. Oper. Res.*, vol. 83, no. 1, pp. 220–236, May 1995. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/037722179590601A>
- [38] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. ICML*, vol. 28, Apr. 2013, pp. 1139–1147.
- [39] B. O'Donoghue and E. Candès, "Adaptive restart for accelerated gradient schemes," *Found. Comput. Math.*, vol. 15, no. 3, pp. 715–732, Jun. 2015.
- [40] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, *Segment Routing Architecture*, document RFC 8402, Jul. 2018.
- [41] P. Kumar et al., "Semi-oblivious traffic engineering: The road not taken," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 157–170.
- [42] N. Geng et al., "Distributed and adaptive traffic engineering with deep reinforcement learning," in *Proc. IEEE/ACM 29th Int. Symp. Quality Service (IWQOS)*, Jun. 2021, pp. 1–10.
- [43] A. Viswanathan, E. C. Rosen, and R. Callon, *Multiprotocol Label Switching Architecture*, document RFC 3031, Jan. 2001.
- [44] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2181–2206, 4th Quart., 2014.
- [45] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, 2006.
- [46] R. Hartert et al., "A declarative and expressive approach to control forwarding paths in carrier-grade networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 15–28, 2015.
- [47] R. Srikant, *The Mathematics of Internet Congestion Control*. Boston, MA, USA: Birkhäuser, 2004.



Ying Tian received the B.E. degree in computer science from Beijing University of Posts and Telecommunications, China, in 2018, and the Ph.D. degree in computer from Tsinghua University in 2023. She is currently an Assistant Researcher with Zhongguancun Laboratory, Beijing, China. Her research interests include traffic engineering, routing optimization, segment routing, and software defined networking.



Zhiliang Wang (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science from Tsinghua University, China, in 2001, 2003, and 2006, respectively. Currently, he is an Associate Professor with the Institute for Network Sciences and Cyberspace and Beijing National Research Center for Information Science and Technology, Tsinghua University, and Zhongguancun Laboratory. His research interests include formal methods and protocol testing, next generation internet, and network measurement.



Xia Yin (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science from Tsinghua University in 1995, 1997, and 2000, respectively. She is currently a Full Professor with the Department of Computer Science and Technology and Beijing National Research Center for Information Science and Technology, Tsinghua University. She is also a Professor with Zhongguancun Laboratory. Her research interests include future internet architecture, formal method, protocol testing, and large-scale internet routing.



Xingang Shi (Member, IEEE) received the B.S. degree from Tsinghua University and the Ph.D. degree from The Chinese University of Hong Kong. He is currently working with the Institute for Network Sciences and Cyberspace and Beijing National Research Center for Information Science and Technology, Tsinghua University. He is also a Researcher with Beijing National Research Center for Information Science and Technology and Zhongguancun Laboratory. His research interests include network measurement and routing protocols.



Jiahai Yang (Senior Member, IEEE) received the B.Sc. degree in computer science from Beijing Technology and Business University and the M.Sc. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China. He is currently a Full Professor with the Institute for Network Sciences and Cyberspace and Beijing National Research Center for Information Science and Technology, Tsinghua University. He is also a Professor with Zhongguancun Laboratory. His research interests include network management, network measurement, network security, internet routing, cloud computing, and big data applications.



Han Zhang received the B.E. degree in computer science and technology from Jilin University and the Ph.D. degree from Tsinghua University. He is currently an Assistant Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. He is also a Professor with Zhongguancun Laboratory. His research interests include computer networks, network security, and network systems.