Learning-NUM: Network Utility Maximization with Unknown Utility Functions and Queueing Delay

Xinzhe Fu
LIDS, Massachusetts Institute of Technology
USA

Eytan Modiano
LIDS, Massachusetts Institute of Technology
USA

ABSTRACT

Network Utility Maximization (NUM) studies the problems of allocating traffic rates to network users in order to maximize the users' total utility subject to network resource constraints. In this paper, we propose a new NUM framework, Learning-NUM, where the users' utility functions are unknown apriori and the utility function values of the traffic rates can be observed only after the corresponding traffic is delivered to the destination, which means that the utility feedback experiences *queueing delay*. The goal is to design a policy that gradually learns the utility functions and makes rate allocation and network scheduling/routing decisions so as to maximize the total utility obtained over a finite time horizon T. In addition to unknown utility functions and stochastic constraints, a central challenge of our problem lies in the queueing delay of the observations, which may be unbounded and depends on the decisions of the policy. We first show that the expected total utility obtained by the best dynamic policy is upper bounded by the solution to a static optimization problem. Without the presence of feedback delay, we design an algorithm based on the ideas of gradient estimation and Max-Weight scheduling. To handle the feedback delay, we embed the algorithm in a parallel-instance paradigm to form a policy that achieves $\tilde{O}(T^{3/4})$-regret, i.e., the difference between the expected utility obtained by the best dynamic policy and our policy is in $\tilde{O}(T^{3/4})$. Finally, to demonstrate the practical applicability of the Learning-NUM framework, we apply it to three application scenarios including database query, job scheduling and video streaming. We further conduct simulations on the job scheduling application to evaluate the empirical performance of our policy.

CCS CONCEPTS

• **Networks** → **Network algorithms**.

ACM Reference Format:
Xinzhe Fu and Eytan Modiano. 2021. Learning-NUM: Network Utility Maximization with Unknown Utility Functions and Queueing Delay. In *The Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '21), July 26–29, 2021, Shanghai, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3466772.3467031

MobiHoc '21, July 26–29, 2021, Shanghai, China

ACM ISBN 978-1-4503-8558-9/21/07.
https://doi.org/10.1145/3466772.3467031

1 INTRODUCTION

Network Utility Maximization (NUM) has been a central problem in networking research for decades and has become a standard framework for making intelligent network resource allocation decisions. It has found a wide range of applications such as congestion control in the Internet [1–3], power allocation in wireless networks [4] and job scheduling in cloud computing [5, 6].

As a network optimization paradigm, NUM studies the problems of user traffic admission control to maximize the users' total utility subject to network resource constraints. Previous works in NUM can be classified into two categories: static and stochastic. In the static approach [1–3, 7, 8], the traffic rates are modeled as flow variables, the bandwidth constraints are modeled as network flow constraints, and the analysis focuses on the convergence rates of the optimization algorithms. In the stochastic approach [4–6, 9, 10], the traffic rates are determined by the time-average admitted traffic, the resource constraints are captured by the long-term stability of the stochastic queueing networks and the analysis focuses on the tradeoff between the long-term average utility and queue length.

Regardless of the differences in modeling and analysis, previous NUM results rest on a key assumption that the utility functions of network users are known. This is justified when the utility functions are simply optimization proxies for network performance criteria such as fairness [1, 2, 9]. However, when the utility represents more concrete quantities such as power and energy consumption [4, 10], user satisfaction [12, 13] and job quality [5, 6, 14], often we do not have prior knowledge of the utility functions, i.e., the functional relationship between the traffic rate of a user and its corresponding utility value is unknown in advance.

In this paper, we propose a new NUM framework, Learning-NUM (L-NUM), where the utility functions are unknown but their values can be learned over the process of decision making. Specifically, we consider a time-varying stochastic queueing network in discrete time, which captures both wireline and wireless networks. There are K users, where user k has a concave utility function f_k that is initially unknown to the network operator. Each user has a corresponding source-destination pair in the network. At every time t, for each user k, the network operator injects a "job" of size $r_k(t)$ from the user's source to be delivered to the user's destination. The job size in our framework resembles the admitted traffic rate in the traditional NUM formulation. We will explain the connection between the two notions in Section 2. Next, the operator chooses a network action that controls the routing and scheduling, which further determines the queue dynamics of the network. Finally, the utility value $(f_k(r_k))$ of a job (of size r_k) can only be observed after the job gets delivered to the destination as feedback from the user.

We study the problem of designing a policy that jointly determines the job sizes and network actions based on the utility function

values learned from observations. We define the utility achieved by a policy as the total utility of the jobs delivered by a finite time horizon T. This definition naturally enforces network resource constraints as the undelivered jobs in the queues at time T are not counted towards the utility. We seek to design a policy with regret sublinear to T, where regret [11] is defined as the gap between the expected utility of the policy and that of the optimal policy that has full knowledge of the utility functions in advance. As a first step, we establish that the expected utility achieved by the optimal (dynamic) policy is upper bounded by T times the optimal value of a static optimization problem, whose objective is the sum of the (unknown) utility functions and the constraints are implicitly given by the capacity region of the network. This result provides an important insight that a policy achieves low regret if it can closely track the solution to the static optimization problem.

While solving an optimization problem with unknown objective function is a common challenge faced in the online convex optimization literature [20, 21], our problem is further complicated by the facts that the constraints, which essentially enforce network stability, are stochastic and unknown in advance (See Section 3 for details). Thus, they cannot be handled by techniques in online optimization that require the feasibility region to be known in advance [21]. Moreover, the utility value can be observed only after the delivery of the job, which, in a First-In-First-Out network, happens after the delivery of the jobs injected before it. This means that the feedback in our problem experiences **queueing-style delay** that may be unbounded and depends on the decisions of the policy. Such delay evades existing techniques in the literature as they typically assume bounded or decision-independent delay [25, 26].

To deal with unknown utility functions and stochastic constraints, we combine the ideas of gradient sampling [15] and max-weight scheduling (back-pressure routing) [16] to propose an online scheduling algorithm that works for the L-NUM problem without feedback delay. We next embed the algorithm into a parallel-instance paradigm to obtain a scheduling policy that can handle the queueing-style feedback delay and achieve $\tilde{O}(T^{3/4})$-regret[1]. Finally, we show how to apply our framework to applications including database query [18], job scheduling [19] and video streaming [12, 13]. We further empirically evaluate the performance of the proposed policy through simulations on job scheduling scenarios.

The rest of the paper is organized as follows. The model and formal definitions of the L-NUM framework are presented in Section 2. In Section 3, we prove the upper bound on the optimal expected utility. In Section 4, we propose the online scheduling algorithm and the parallel-instance paradigm for the L-NUM framework. We further illustrate several applications of L-NUM in Section 5. The empirical performance of the online scheduling policy is evaluated in Section 6. Finally, we conclude the paper with some future directions in Section 7.

2 MODEL AND PROBLEM FORMULATION

In this section, we specify the general network model and set up the framework of network utility maximization with unknown utility functions. We consider a network $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ being the set of nodes and $\mathcal{E}$ being the set of directed links. For each node $i \in \mathcal{V}$, we will denote its set of outgoing neighbors by $\mathcal{N}_i$. There are K classes of users in the network. Each user k corresponds to a job class (also denoted by k), and is mapped to one source-destination pair (s_k, d_k). Multiple job classes can be mapped to the same source-destination pair. Source s_k sends class-k jobs that get delivered to d_k through the network. We will refer to the jobs sent from s_k destined to d_k as class-k traffic. Each node $i \in \mathcal{V}$ has a queue Q_i^k that buffers the incoming class-k traffic of node i. The network operates in discrete time $t = 1, \ldots, T$, where T is the specified time horizon. At each time t, the network is in state $\omega(t) \in \mathcal{W}$, with $\mathcal{W}$ denoting the set of possible network states. In concrete applications, the network states may correspond to channel states of links, service states of servers, or simply a placeholder when the network is static with only one state. We assume $\omega(t)$'s is a sequence of i.i.d. random element with $\mathbb{P}(\omega(t) = \omega) = p(\omega)$. However, the distribution of $\omega(t)$ is unknown to the network operator.

2.1 Traffic Model and Network Dynamics

At each time t, the network operator first observes the current network state $\omega(t)$. It next chooses job size $r_k(t)$ for each class and sends a job of size $r_k(t)$ to the buffer $Q_{s_k}^k$, where $r_k(t)$ is a real value that satisfies $0 \le r_k(t) \le B$. The job size corresponds to the amount of admitted traffic at a time slot. For example, as we will demonstrate in Section 5, in video streaming, the job size represents the resolution of a video chunk sent to the user. We adopt this discrete notion of job size rather than the continuous notion of traffic rate in the traditional NUM framework because job size is more suitable for our finite-horizon discrete-time framework. Finally, the network operator chooses a network action $\boldsymbol{x}(t) \in \mathcal{X}$ that incorporates the routing and scheduling decisions of the network. The feasible set of actions $\mathcal{X}$ can be discrete or continuous. For each $\boldsymbol{x} \in \mathcal{X}$, under network state ω, we use $A_{ij}^k(\omega, \boldsymbol{x})$ to denote the offered transmission rate on link (i, j) for class k, i.e., the amount of class-k traffic that can be sent from node i to node j. Each link transmits traffic in a First-In-First-Out (FIFO) basis. $A_{ij}^k(\omega, \boldsymbol{x})$'s are assumed to be non-negative and upper bounded by A for all ω and $\boldsymbol{x}$. Based on the definitions above, the dynamics of the queues can be written following the Lindley recursion:

$$Q_{s_k}^k(t+1) = [Q_{s_k}^k(t) + r_k(t) - \sum_{j \in \mathcal{N}_{s_k}} A_{s_k j}^k(\omega(t), \boldsymbol{x}(t))]^+, \forall k, \tag{1}$$

$$Q_{d_k}^k(t) = 0, \quad \forall k, \tag{2}$$

$$Q_i^k(t+1) = [Q_i^k(t) + \sum_{j:i \in \mathcal{N}_j} A_{ji}^k(\omega(t), \boldsymbol{x}(t)) - \sum_{j \in \mathcal{N}_i} A_{ij}^k(\omega(t), \boldsymbol{x}(t))]^+, \forall i \ne s_k, d_k. \tag{3}$$

We define $\Lambda(\omega) := \{(A(\omega, \boldsymbol{x}))_{ij}^k, \boldsymbol{x} \in \mathcal{X}\}$ as the set of feasible transmission rate vectors under network state ω. Note that the network operator can observe $\Lambda(\omega(t))$ at t but does not know the distribution of $\omega(t)$. We assume that $\Lambda(\omega)$ is downward closing. Finally, we define $Cap(\mathcal{G})$ as the set of feasible rate vectors $(r_1, \ldots, r_K)$, i.e., there exists $\{\lambda(\omega)\}_{ij}^k \in Conv(\Lambda(\omega))$ with

$$\forall k,\ r_k \le \sum_{\omega \in \mathcal{W}} \sum_{j \in \mathcal{N}_{s_k}} p(\omega) \lambda(\omega)_{ij}^k,$$

$$\forall i \in \mathcal{V},\ \sum_{\omega \in \mathcal{W}} \sum_{j:i \in \mathcal{N}_j} p(\omega) \lambda(\omega)_{ji}^k \le \sum_{\omega \in \mathcal{W}} \sum_{j \in \mathcal{N}_i} p(\omega) \lambda(\omega)_{ij}^k,$$

[1] $\tilde{O}(\cdot)$ hides logarithmic factors of T.

where $conv(\Lambda(\omega))$ is the convex hull of $\Lambda(\omega)$. $Cap(\mathcal{G})$ resembles the network capacity region in the traditional infinite-horizon network utility maximization problem [9], i.e., the set of traffic rate vectors that can be supported by the network. However, as we consider a finite-horizon setting here, the capacity region here does not exactly characterize the set of job-size vectors that the network can support. Nevertheless, the close connection between the two concepts will be revealed in **Section 3**. Furthermore, to prevent trivializing the problem, we enforce the condition on $Cap(\mathcal{G})$ that it has non-empty interior, i.e., there exists $\eta > 0$ such that $(\eta, \ldots, \eta) \in Cap(\mathcal{G})$.

2.2 Utility Model

Each job class (user) k is associated with some underlying utility function f_k. The utility functions are initially unknown. When a class-k job of size r_k gets delivered to d_k, we observe and obtain utility of value $f_k(r_k)$. Note that this implies that the utility feedback of each job experiences queueing-style delay, i.e., the time from injecting a job into the network to observing its utility value is equal to the time that the job spends in the network (queues). See Figure 1 for further illustration of the feedback delay.

For each traffic class k, we assume its underlying utility function has the following properties:

(1) f_k is monotonically non-decreasing and concave.
(2) f_k is bounded on $[0, B]$, i.e., $\forall r \in [0, B]$, $f_k(r) \leq D$ for some constant D.
(3) f_k is L-Lipschitz continuous, i.e., $\forall r_1, r_2 \in [0, B]$, $|f_k(r_2) - f_k(r_1)| \leq L \cdot |r_2 - r_1|$.

2.3 Problem Formulation

Given the network $\mathcal{G}$ and time-horizon T, we seek to find a scheduling policy that determines the sizes of the jobs sent by the sources and the network actions that maximizes the total utility obtained at the end of the horizon T. Formally, let Π be the collection of *admissible policies* that make scheduling decisions at time t based on observations obtained before time t. Policies in Π do not have access to the underlying utility functions or the distribution of network state, but can learn them through observations of utility values and instantiated network state. We further let $\bar{\Pi}$ be the collection of all policies, including non-admissible policies that know the underlying utility functions and the network state distribution. For a policy π, we define $U(\pi, T)$ to be the total utility obtained from jobs that are delivered by time T under π. Note that $U(\pi, T)$ is a random variables, the randomness of which comes from the time-varying network state and the (possible) inherent randomness of the scheduling policy. We adopt the notion of regret from the online learning literature as the measure of quality of scheduling policies.

Definition 1 (Regret). *The regret of scheduling policy π is defined as*

$$R(\pi, T) = \sup_{\pi^* \in \bar{\Pi}} \mathbb{E}[U(\pi^*, T)] - \mathbb{E}[U(\pi, T)],$$

The regret $R(\pi, T)$ measures the gap between the expected utility obtained under π and the maximum utility achieved by any (even non-admissible) policy for the given instance.

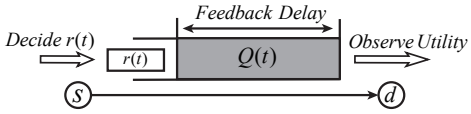

Figure 1: A single-queue example illustrating the queueing-style feedback delay in the L-NUM framework.

Based on the above preliminaries, we formally pose the problem of network utility maximization with unknown utility functions, which we will refer to as the L-NUM (Learning-NUM) problem, as one that asks for an admissible scheduling policy with low regret.

Definition 2 (The L-NUM Problem). *The L-NUM problem seeks an admissible policy π with sublinear regret, i.e., $\lim_{T\to\infty} \frac{R(\pi,T)}{T} = 0$.*

Remark: (i). A policy that has sublinear regret is *asymptotic optimal*, since the gap between time-average utility achieved by the policy and that of the optimal goes to zero. (ii). Although the regret does not explicitly depend on the queue backlogs at the end of the horizon T, the queue backlogs are implicitly accounted for, since the utility $U(\pi, T)$ does not include the jobs that are still in the queue at time T.

3 UPPER BOUND ON THE OPTIMAL UTILITY

If the utility functions are known in advance, L-NUM becomes a finite-horizon stochastic optimization problem. Typically, the optimal policy for the problem is a dynamic programming-based policy that is intractable and difficult to compare to. Therefore, in this section, we relate the expected utility obtained by the best policy in $\bar{\Pi}$ to the optimal value of a static optimization problem, which motivates the design and analysis of the admissible scheduling policy we propose. The optimization problem $\mathcal{P}$ is defined as follows:

$$\mathcal{P}: \max_{\{r\}_k} \sum_{k=1}^{K} f_k(r_k) \tag{4}$$

$$\textbf{s.t.} \quad (r_1, \ldots, r_K) \in Cap(\mathcal{G}) \tag{5}$$

$$r_k \in [0, B], \quad \forall k. \tag{6}$$

Intuitively, the optimization problem characterizes a static version of the L-NUM problem over the job-size variables. The decision variables $\{r_k\}$'s can be interpreted as average size of jobs of class k. $\mathcal{P}$ seeks to maximize the total utility obtained by $\{r_k\}$ such that the vector lies inside the network capacity region. Note that $Cap(\mathcal{G})$ is a convex set over $\{r_k\}$. Hence, $\mathcal{P}$ is a convex optimization problem.

Based on the optimization problem $\mathcal{P}$, we are ready to state the main result of this section, i.e., the optimal value of $\mathcal{P}$ multiplied by the time horizon upper-bounds the maximum expected utility over all policies in $\bar{\Pi}$.

Theorem 1. $\sup_{\pi^* \in \bar{\Pi}} \mathbb{E}[U(\pi^*, T)] \leq T \cdot OPT(\mathcal{P})$.

Proof. The main idea of the proof is that, for any given policy, we first take certain averages of the job sizes of each traffic class and then show that the averages satisfy the constraints of $\mathcal{P}$. Next, by the concavity of the underlying utility functions, their corresponding value of the objective function is no less than the expected utility of the policy.

For ease of notations, we prove the theorem for deterministic policies in $\bar{\Pi}$. The case of randomized policies follows similarly.

Consider an arbitrary policy $\pi^* \in \bar{\Pi}$ and a sample path θ of its execution on the problem instance. For each traffic class k, let $r_k(1,\theta),\ldots,r_k(T,\theta)$ be the size of the jobs specified by π^* over the time horizon T. Define $\tilde{r}_k(t,\theta) = r_k(t,\theta)$ if the t-th job is delivered by time T and $\tilde{r}_k(t,\theta) = 0$ otherwise. Let $\boldsymbol{x}(t,\theta)$ be the network action chosen by π^* at t and let $\omega(t,\theta)$ be the network state at t under θ. Based on the utility model we have that the utility achieved by π^* on sample path θ is equal to $\sum_{k=1}^{K}\sum_{t=1}^{T} f_k(\tilde{r}_k(t,\theta))$. Let $\bar{r}_k(\theta) = \frac{1}{T}\sum_{t=1}^{T}\tilde{r}_k(t,\theta)$. Since the underlying utility functions are concave, we have

$$\sum_{k=1}^{K}\sum_{t=1}^{T} f_k(\tilde{r}_k(t,\theta)) \le T\sum_{k=1}^{K} f_k(\bar{r}(\theta)). \tag{7}$$

Furthermore, let $\tilde{A}_{ij}^k(\omega(t,\theta),\boldsymbol{x}(t,\theta))$ be the realized transmission rate on link (i,j) for class-k at t. The realized transmission $\tilde{A}_{ij}^k$ is equal to the offered transmission A_{ij}^k when the queue length is greater than the offered transmission, and the realized transmission is smaller otherwise. From the queue dynamics (Equations 1, 2 and 3), we obtain that

$$\forall k,\quad T\bar{r}_k(\theta) \le \sum_{t=1}^{T}\sum_{j\in\mathcal{N}_{s_k}} \tilde{A}_{s_k j}^k(\omega(t,\theta),\boldsymbol{x}(t,\theta)), \tag{8}$$

$$\forall i \ne s_k, d_k, \sum_{t=1}^{T}\sum_{j:i\in\mathcal{N}_j} \tilde{A}_{ji}^k(\omega(t,\theta),\boldsymbol{x}(t,\theta)) \le \sum_{t=1}^{T}\sum_{j\in\mathcal{N}_i} \tilde{A}_{ij}^k(\omega(t,\theta),\boldsymbol{x}(t,\theta)). \tag{9}$$

Define $\hat{p}_\theta(\omega), \omega\in\mathcal{W}$ as the empirical distribution of ω,

$$\hat{p}_\theta(\omega) := \frac{\sum_{t=1}^{T}\mathbb{1}\{\omega(t,\theta)=\omega\}}{T}.$$

It follows from (8), (9) that for each $\omega\in\mathcal{W}$, there exists $(\tilde{\lambda}(\omega,\theta))_{ij}^k \in Conv(\Lambda(\omega))$ such that

$$\forall k,\quad \bar{r}_k(\theta) \le \sum_{\omega\in\mathcal{W}}\sum_{j\in\mathcal{N}_{s_n}} \hat{p}_\theta(\omega)\tilde{\lambda}_{s_k j}^k(\omega,\theta),$$

$$\forall i\ne s_k,d_k,\ \sum_{\omega\in\mathcal{W}}\sum_{j:i\in\mathcal{N}_j} \hat{p}_\theta(\omega)\tilde{\lambda}_{ji}^k(\omega,\theta) \le \sum_{\omega\in\mathcal{W}}\sum_{j\in\mathcal{N}_i} \hat{p}_\theta(\omega)\tilde{\lambda}_{ij}^k(\omega,\theta).$$

Moreover, as $\Lambda(\omega)$ is downward-closing, we further have that there exists $(\lambda(\omega,\theta))_{ij}^k \in Conv(\Lambda(\omega))$ such that

$$\forall k,\quad \bar{r}_k(\theta) = \sum_{\omega\in\mathcal{W}}\sum_{j\in\mathcal{N}_{s_k}} \hat{p}_\theta(\omega)\lambda_{s_k j}^k(\omega,\theta),$$

$$\forall i\ne s_k,d_k,\quad \sum_{\omega\in\mathcal{W}}\sum_{j:i\in\mathcal{N}_j} \hat{p}_\theta(\omega)\lambda_{ji}^k(\omega,\theta) = \sum_{\omega\in\mathcal{W}}\sum_{j\in\mathcal{N}_i} \hat{p}_\theta(\omega)\lambda_{ij}^k(\omega,\theta).$$

Taking expectation over θ, we have $(\mathbb{E}_\theta[\bar{r}_1(\theta)],\ldots,\mathbb{E}_\theta[\bar{r}_1(\theta)]) \in Cap(\mathcal{G})$. Moreover, it is easy to see that $0 \le \mathbb{E}_\theta[\bar{r}_k(\theta)] \le B$ for all k. Therefore, the vector $(\mathbb{E}_\theta[\bar{r}_1(\theta)],\ldots,\mathbb{E}_\theta[\bar{r}_1(\theta)])$ is feasible to $\mathcal{P}$. Hence, $OPT(\mathcal{P}) \ge \sum_{k=1}^{K} f_k(\mathbb{E}_\theta[\bar{r}_k(\theta)])$. Invoking the concavity of f_k's again, by Jensen's inequality, we have for all k, $f_k(\mathbb{E}_\theta[\bar{r}_k(\theta)]) \ge \mathbb{E}_\theta[f_k(\bar{r}_k(\theta))]$. Combining this with (7), we obtain

$$\begin{aligned} OPT(\mathcal{P}) &\ge \sum_{k=1}^{K} f_k(\mathbb{E}_\theta[\bar{r}_k(\theta)]) \ge \mathbb{E}\left[\sum_{k=1}^{K} f_k(\bar{r}_k(\theta))\right] \\ &\ge \frac{1}{T}\mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{T} f_k(\tilde{r}_k(t,\theta))\right], \end{aligned} \tag{10}$$

which concludes the proof. □

It is worth pointing out that Theorem 1 does not imply that the optimal policy is a static one that assigns the job sizes according to the solution to the optimization problem $\mathcal{P}$. Such a policy would not achieve an expected utility of $T\cdot OPT$ since the expected number of jobs delivered is typically less than T. Indeed, due to the stochastic network dynamics, a portion of the jobs will still remain in the queues by the end of the time horizon. Despite that, the theorem does provide the insight that a policy achieves low regret if it can closely approximate the solution to $\mathcal{P}$ at each time slot. As the objective function of $\mathcal{P}$ is unknown, the problem has similar flavor to online/zeroth-order optimization [20, 23, 24]. However, in the L-NUM problem we are facing two additional challenges. First, the feasibility region in the L-NUM is stochastic and not explicitly given as the distribution of network states is unknown. Thus, we cannot rely on method that requires the feasibility region to be known in advance [21]. Second, the queueing-style delay of the feedback compromises the policy's ability to adjust based on utility observations. As the delay is action-dependent and may be unbounded, it also poses more stringent requirement on controlling the network queue lengths.

4 ONLINE SCHEDULING POLICY

In this section, we introduce the scheduling policy we propose for the L-NUM framework – the Parallel Gradient Sampling Max-Weight (P-GSMW) policy. The P-GSMW policy is composed of embedding an algorithm (called Gradient Sampling Max-Weight, GSMW) that makes job-size and scheduling decisions based on immediate feedback (no delay) into a parallel-instance paradigm that handles the feedback delay. The GSMW algorithm essentially combines the ideas of drift-plus-penalty optimization [17], gradient sampling [15], and Max-Weight scheduling. The parallel-instance paradigm invokes multiple parallel instances of the GSMW algorithm such that each instance essentially runs in a no-delay setting. In the following, we first introduce the GSMW algorithm, and then combine it with the parallel-instance paradigm. Finally, we provide discussion on the challenges posed by the feedback delay.

4.1 The GSMW Algorithm

In the presentation of the GSMW algorithm, we assume a no-delay setting, i.e., the utility values of the jobs can be observed immediately after job-size decision. We will handle the feedback delay with the parallel-instance paradigm in subsequent sections.

The GSMW algorithm (**Algoritm 1**) maintains a virtual job size variable $\hat{r}_k$ for each class k and utilizes queue lengths to update the virtual job size variables and network actions. The $\hat{r}_k$'s are updated once every two slots, which essentially divides the time horizon

into epochs of size two (without loss of generality, we assume the horizon T to be even). For simplicity of notations, we will assume that the network state remains unchanged for each epoch and refer to an epoch as a time slot indexed by $t \in \{1, \ldots, T\}$ for the rest of the paper, i.e., at each slot, we need to make scheduling decision and job-size decision for two incoming jobs of each class.[2]

At each slot $t \in \{1, \ldots, T\}$, the network action is chosen according to a Max-Weight-like rule (Line 3). The decisions on job size are made based on the virtual job size variables at the corresponding epoch. The updates of virtual job size variables are determined by gradient estimates of the utility functions and queue lengths. Since the utility functions are unknown, GSMW constructs the gradient estimates using observations of function values. Specifically, at slot t, each source s_k injects a first job of size $\hat{r}_k(\tau) + \delta$ and a second job of size $\hat{r}_k(\tau) - \delta$ for each $k \in n$ and obtains the feedback of value $f_k(\hat{r}_k(\tau) + \delta)$ and $f_k(\hat{r}_k(\tau) - \delta)$ (Lines 5, 6). The two feedback values obtained are combined to form the gradient estimate of f_k at $\hat{r}_k(\tau)$ (Line 7). The gradient estimate is then fed into the update of the virtual variable $\hat{r}_k$ (Line 10). The projection step $\mathcal{P}_{[\delta, B-\delta]}$ of Line 10, defined as the projection on to interval $[\delta, B - \delta]$ by the Euclidean norm, is to ensure that $\hat{r}_k(\tau) + \delta$ and $\hat{r}_k(\tau) - \delta$ always lie in the domain $[0, B]$. Here, parameter V controls the relative weights of gradient and queue length while parameter α determines the step size.

Algorithm 1 The Gradient Sampling Max-Weight Algorithm

Input: Network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, parameters V, δ, α

1: **Initialize:** $\boldsymbol{x}(0) \in \mathcal{X}, \hat{r}_k(0) = \delta$.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $\quad \boldsymbol{x}(t) := \arg\max_{\boldsymbol{x} \in \mathcal{X}} \sum_{i \in V} \sum_{k=1}^{K} A_{ij}^k(\omega(t), \boldsymbol{x}) [Q_i^k(t) - Q_j^k(t)]$
4: $\quad$ **for** $k = 1, \ldots, K$ **do**
5: $\quad\quad s_k$ injects job of size $\hat{r}_k(t) + \delta$ and observes $f_k(\hat{r}_k(t) + \delta)$.
6: $\quad\quad s_k$ injects job of size $\hat{r}_k(t) - \delta$ and observes $f_k(\hat{r}_k(t) - \delta)$.
7: $\quad\quad \hat{\nabla} f_k(\hat{r}_k(t)) := \frac{f_k(\hat{r}_k(t)+\delta) - f_k(\hat{r}_k(t)-\delta)}{2\delta}$
8: $\quad$ Update queue lengths according to $r_k(t), \boldsymbol{x}(t)$.
9: $\quad$ **for** $k = 1, \ldots, K$ **do**
10: $\quad\quad \hat{r}_k(t+1) := \mathcal{P}_{[\delta, B-\delta]} \left[\hat{r}_k(t) + \frac{1}{\alpha} (V \cdot \hat{\nabla} f_k(\hat{r}_k(t)) - Q_{s_k}^k(t)) \right]$

4.2 The Parallel-Instance Paradigm

In order to handle the feedback delay, we design a parallel-instance paradigm that encapsulates the GSMW algorithm, and forms the Parallel-instance GSMW (P-GSMW) policy. The details of the P-GSMW policy are shown in **Algorithm 2**. Similar to the GSMW algorithm, the network action $\boldsymbol{x}(t)$ at each time slot is still determined by the Max-Weight rule. The key difference is that the paradigm maintains a set of parallel instances of the GSMW algorithm, which we will refer to as the instance reservoir $\mathcal{I}$. Each instance can be in one of the two possible status: FRESH and STALE. FRESH status means that the instance has obtained the corresponding utility feedback and can perform updates on the virtual job-size variables (Line 5 of **Algorithm 2**); STALE status means that the instance is still waiting for utility feedback. We use $\{\hat{r}_k^I(t)\}$ to denote the virtual job size variables maintained by instance I. When we need to make job size decisions, if there is a FRESH instance available in the reservoir, we "invoke" the instance by performing updates and deciding on the job sizes based on the updated virtual job-size variables of the instance (Line 5 of **Algorithm 2**). If there are multiple FRESH instances, we select one arbitrarily. We then change the instance's status to STALE (Line 7). If there is no FRESH instance available, we initialize a new instance, add it to $\mathcal{I}$ (Lines 9 and 10). The virtual job-size variables of instances that are not invoked remains unchanged (Line 12). Upon delivery of jobs, we observe utility values and feed them to the corresponding instances. If an instance has all the utility observations available for the jobs injected when it was last invoked, we change its status from "STALE" to "FRESH" (Line 15). We further illustrate the parallel-instance paradigm with a single-queue example in Figure 2.

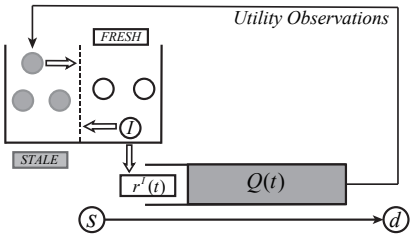

Figure 2: A single-queue example illustrating the parallel-instance paradigm.

Remark: Our parallel-instance paradigm has a similar flavor to the technique in [25] for online learning with delayed feedback. However, the observation delay in the L-NUM framework may be unbounded and is action-dependent, which is more general than the bounded, decision-independent delay considered in [25].

4.3 Policy Analysis

In this section, we analyze the regret achieved by the P-GSMW policy π_{P-GSMW}. The main result is presented Theorem 2. Due to space limitation, we will only provide proof sketches, and defer the full proofs to our technical report [27].

Theorem 2. *The Parallel-instance Gradient Sampling Max-Weight policy π_{P-GSMW} achieves $\tilde{O}(T^{3/4})$ regret by setting $\alpha = 2K\sqrt{T}/\eta, V = T^{1/4}, \delta = T^{-1/2}$., i.e.,*

$$R(\pi_{P-GSMW}, T) = \sup_{\pi^* \in \bar{\Pi}} \mathbb{E}[U(\pi^*, T)] - \mathbb{E}[U(\pi_{P-GSMW}, T)] = \tilde{O}(T^{3/4}).$$

Remark: It can be shown that without feedback delay, by setting $\alpha = O(T), V = O(\sqrt{T}), \delta = O(1/\sqrt{T})$ (rather than $\alpha = O(\sqrt{T}), V = O(T^{1/4}), \delta = O(1/\sqrt{T})$ in Theorem 2), the GSMW algorithm achieves a regret of order $\tilde{O}(\sqrt{T})$, which matches the established regret lower bound $\Omega(\sqrt{T})$ [21]. Under the queueing-style feedback delay, the P-GSMW policy achieves $\tilde{O}(T^{3/4})$ which is higher than $\tilde{O}(\sqrt{T})$. This raises the question whether the delay of L-NUM fundamental increases the difficulty of the problem, i.e., a lower bound better than $\Omega(\sqrt{T})$ can be shown, or that there exists algorithm for L-NUM that has regret better than $\tilde{O}(T^{3/4})$. We leave this as a future direction.

[2]This assumption is purely made for notational convenience. Our results can be straightforwardly adapted to the original setting without the assumption.

Algorithm 2 The Parallel-instance GSMW Policy

Input: Network $\mathcal{G}(\mathcal{V},\mathcal{E})$, parameters V,δ,α, instance reservoir $\mathcal{I}$

1: **for** $t=1,2,\ldots,T$ **do**
2: $\boldsymbol{x}(t) \coloneqq \arg\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{i\in V}\sum_{k=1}^{K} A_{ij}^{k}(\omega(t),\boldsymbol{x})[Q_i^k(t)-Q_j^k(t)]$
3: **if** There exists a FRESH instance $I_t\in\mathcal{I}$ **then**
4: **for** $k=1,\ldots,K$ **do**
5: $\hat{r}_k^{I_t}(t) \coloneqq \mathcal{P}_{[\delta,B-\delta]}\left[\hat{r}_k^{I_t}(t-1)+\frac{1}{\alpha}(V\cdot\hat{\nabla}f_k(\hat{r}_k^{I_t}(t-1))-Q_{s_k}^k(t))\right]$
6: s_k injects one job of size $\hat{r}_k^{I_t}(t)+\delta$ and another job of size $\hat{r}_k^{I_t}(t)-\delta$.
7: Change the status of I_t to STALE.
8: **else**
9: Create a new instance I_t # No FRESH instance in $\mathcal{I}$.
10: For each k, initialize $\hat{r}_k^{I_t}(t)\coloneqq\delta$, and s_k injects job of size $\hat{r}_k^{I_t}(t)+\delta$ and another job of size $\hat{r}_k^{I_t}(t)-\delta$
11: Update queue lengths according to $r_k(t),\boldsymbol{x}(t)$.
12: $\{\hat{r}_k^{J}(t)\}\coloneqq\{\hat{r}_k^{J}(t)\}$ for $J\in\mathcal{I}, J\neq I_t$.
13: Collect utility observations from delivered jobs and form gradient estimates $\hat{\nabla}f_k(\hat{r}_k^I(t)) \coloneqq \frac{f_k(\hat{r}_k^I(t)+\delta)-f_k(\hat{r}_k^I(t)-\delta)}{2\delta}$
14: **for** STALE instance $I\in\mathcal{I}$ **do**
15: Change the status of I to FRESH if it has obtained all outstanding gradient estimates.

Proof. (of Theorem 2) As we focus on bounding the regret with respect to the time horizon T, we will use C to represent a generic constant that does not depend on T. Note that C may depend on parameters such as A,B,D,L, and the C's that appear in different equations might not be equal. Instead of directly analyzing the P-GSMW policy, we will analyze the GSMW algorithm (**Algorithm 1**) in a no-delay setting, and illustrate how to extend the analysis to the P-GSMW policy in the end. A complete analysis of the P-GSMW policy can be found in our technical report [27].

Recall that in the no-delay setting, we can observe the utility value immediately after the job-size decision. Thus, we can apply the GSMW algorithm π_{GSMW} with the same parameter values as indicated in Theorem 2. We will show that in this case, the GSMW algorithm achieves $\tilde{O}(T^{3/4})$ regret. We first decompose the regret into two components: one incurred through incrementally solving $\mathcal{P}$ (*Utility Regret*) and the other caused by the undelivered jobs in the queues at the end of the time horizon (*Queueing Regret*).

Lemma 1. *Let $\{r^*\}_k$ be the optimal solution to $\mathcal{P}$,*

$$R(\pi_{GSMW},T) \leq 2\mathbb{E}\left[\sum_{t=1}^{T}\sum_{k=1}^{K} f_k(r_k^*)-f_k(\hat{r}_k(t))\right]+C\sum_{i\in V}\sum_{k=1}^{K}\mathbb{E}[Q_i^k(T)]+CT\delta.$$

Proof Sketch: By definition, the utility achieved by π_{GSMW} is equal to $\sum_{t=1}^{T}\sum_{k=1}^{K} f_k(\hat{r}_k(t)+\delta)+f_k(\hat{r}_k(t)-\delta)$ minus the total utility of (undelivered) jobs in the queue at time T. By Theorem 1 and that the utility of a single job is upper bounded by D, i.e., the upper bound of the utility function, we can bound the regret by the RHS of Lemma 1, where the first term (utility regret) accounts for the cumulative difference between the $\sum_k^K f_k(\hat{r}_k(t))$ and $OPT(\mathcal{P})$, the second term (queueing regret) accounts for the jobs in the queue, and the third term comes from that the sizes of jobs injected by the sources are δ-away from the virtual job size variables.

By taking $\delta=T^{-1/2}$, we have $CT\delta=O(\sqrt{T})$. To prove Theorem 2, we can thus proceed to bound the queueing regret $\sum_{i\in V,k}\mathbb{E}[Q_i^k(T)]$ and the utility regret $\mathbb{E}\left[\sum_{t=1}^{T}\sum_{k=1}^{K} f_k(r_k^*)-f_k(\hat{r}_k(t))\right]$. In the following, we will first show that the total queue length of the network (and thus the queue length regret) under the GSMW algorithm is of order $\tilde{O}(\sqrt{T})$. Using this, we will then show that the utility regret is of order $\tilde{O}(T^{3/4})$.

The GSMW algorithm controls the utility regret and queueing regret through the updates of job-size variables (Line 10 of **Algorithm 1**). The term $V\cdot\hat{\nabla}f_k(\hat{r}_k(t))$ moves the variables towards optimizing utilities, and the $-Q_{s_k}^k(t)$ part aims at controlling queue lengths while α controls the step size. For any $\{r\}_k$ with $r_k\in[\delta,B-\delta]$, by expanding the term $\sum_{k=1}^{K}(\hat{r}_k(t+1)-r_k)^2$ using Line 10, we have the following lemma.

Lemma 2. *For each t, for any $\{r\}_k$ with $r_k\in[\delta,B-\delta]$*

$$\sum_{k=1}^{K}\left[V\hat{\nabla}f_k(\hat{r}_k(t))(r_k-\hat{r}_k(t))\right]+\sum_{k=1}^{K}\left[Q_{s_k}^k(t)\hat{r}_k(t)\right] \leq \sum_{k=1}^{K}\left[Q_{s_k}^k(t)r_k+\alpha[(\hat{r}_k(t)-r_k)^2-(\hat{r}_k(t+1)-r_k)^2]+C.\right]$$

From the Max-Weight selection rule of network action (Line 2 of **Algorithm 2**), we have the following lemma.

Lemma 3. *At every time slot t, for any $\boldsymbol{x}\in\mathcal{X}$, and for all $\{r\}_k$*

$$\begin{aligned}&\sum_{k=1}^{K}Q_{s_k}^k(t)\left[r_k-\sum_{j\in\mathcal{N}_{s_k}}A_{s_kj}^k(\omega(t),\boldsymbol{x}(t))\right]\\&\quad+\sum_{k=1}^{K}\sum_{i\in V,i\neq s_k}Q_i^k(t)\left[\sum_{j:i\in\mathcal{N}_j}A_{ji}^k(\omega(t),\boldsymbol{x}(t))-\sum_{j\in\mathcal{N}_i}A_{ij}^k(\omega(t),\boldsymbol{x}(t))\right]\\&\leq\sum_{k=1}^{K}Q_{s_k}^k(t)\left[r_k-\sum_{j\in\mathcal{N}_{s_k}}A_{s_kj}^k(\omega(t),\boldsymbol{x})\right]\\&\quad+\sum_{k=1}^{K}\sum_{i\in V,i\neq s_k}Q_i^k(t)\left[\sum_{j:i\in\mathcal{N}_j}A_{ji}^k(\omega(t),\boldsymbol{x})-\sum_{j\in\mathcal{N}_i}A_{ij}^k(\omega(t),\boldsymbol{x})\right]\end{aligned}$$

Lemmas 2 and 3 lay the foundation of the analysis of queueing regret and utility regret.

4.3.1 Queueing Regret. We write the vector of queue lengths at time t as $\boldsymbol{Q}(t)$. The key to bounding the queueing regret is to bound the quadratic drift of $\boldsymbol{Q}(t)$. The drift consists of terms involving the queues at the source $\{Q_{s_k}^k\}$, and the queues in the network $\{Q_i^k\}$. We use Lemma 2 to handle the former, and use Lemma 3 for the latter.

Lemma 4. *There exists $\epsilon > 0$ such that under the GSMW algorithm, for all $t \le T$,*

$$\mathbb{E}[||\boldsymbol{Q}(t+1)||^2 - ||\boldsymbol{Q}(t)||^2 \mid \boldsymbol{Q}(t)] \le -\epsilon \sum_{i\in V}\sum_{k=1}^{K} Q_i^k(t) + C\sqrt{T}.$$

Proof Sketch: The drift argument follows from Lemmas 2 and 3 by taking $\{r\}_k$ therein to be the vector $(\delta, \ldots, \delta)$ and utilizing the Slater's condition.

Based on Lemma 4, we use a result on stochastic processes with negative drift from [17], which leads to Proposition 1 that essentially concludes the analysis of the queueing regret.

Proposition 1. *Under GSMW, $\forall t \le T$, $\mathbb{E}[\sum_{i\in V}\sum_{k=1}^{K} Q_i^k(t)] \le \tilde{O}(\sqrt{T})$.*

4.3.2 Utility Regret. It can be shown that the gradient estimate of the GSMW algorithm $\hat{\nabla} f_k$ is equal to the gradient of a smoothed version of f_k defined as $\tilde{f}_k(r) = \frac{1}{2\delta}\int_{-\delta}^{\delta} f_k(r+z)\mathrm{d}z$. Note that by definition $\tilde{f}_k$ is also concave and Lipschitz-continuous. Moreover, by the concavity and Lipschitz continuity of f_k, for all $r \in [0, B]$, $f_k(r) - C\delta \le \tilde{f}_k(r) \le f_k(r)$. Hence, by using $\hat{\nabla} f_k$ as gradients, we are essentially optimizing with respect to objective function $\sum_{k=1}^{K}\tilde{f}_k(r)$, which is at most $C\delta$ away from the true objective function $\sum_{k=1}^{K} f_k(r)$. Accumulating over T time slots, such approximation contributes to at most $O(\sqrt{T})$-regret. Next, from Lemma 2, we take $\{r\}_k$ to be the optimal solution $\{r^*\}_k$ to the optimization problem $\mathcal{P}$ and use $V\hat{\nabla} f_k(\hat{r}_k(t))(r_k^* - \hat{r}_k(t)) \ge V(\tilde{f}_k(r_k^*) - \tilde{f}_k(\hat{r}_k(t)))$. This will us a handle on the utility regret. Finally, we bound the utility regret by summing over the inequality of Lemma 2. Note that the term $\alpha[(\hat{r}_k(t) - r_k)^2 - (\hat{r}_k(t+1) - r_k)^2]$ telescopes, and the terms involving $Q_{s_k}^k(t)$'s can be bounded by previous results on the queueing regret (Proposition 1). Plugging in the parameter values, we can show that the utility regret is of order $\tilde{O}(T^{3/4})$. Combining the analysis of queueing regret, utility regret and Lemma 1 concludes the proof of Theorem 2.

4.3.3 Extension to P-GSMW. The analysis of the P-GSMW policy essentially follows the same vein. Lemmas 1,2 and 3 still hold by replacing $\hat{r}_k(t)$ by $\hat{r}_k^{I_t}(t)$, which is the job-size variables used by the P-GSMW policy at time t. Note that here I_t denotes the instance used at t, and I_t may be different at different t. Using 2 and 3, the analysis of queueing regret can be carried out in a similar way for the P-GSMW policy, as it does not need $\hat{r}_k^{I_t}(t)$'s to come from the same instance at each time but only relies on them being bounded. Therefore, we can establish that under the P-GSMW policy, the queueing regret is still of order $\tilde{O}(\sqrt{T})$.

Proceeding to the utility regret, most of the previous reasoning still holds for the P-GSMW policy, except that the term that corresponds to $\alpha[(\hat{r}_k(t) - r_k)^2 - (\hat{r}_k(t+1) - r_k)^2]$ becomes $\alpha[(\hat{r}_k^{I_t}(t-1) - r_k)^2 - (\hat{r}_k^{I_t}(t) - r_k)^2]$, which no longer telescopes since I_t may change with t. Instead, it only partially telescopes, leading to a term of $O(\alpha|\mathcal{I}|)$ where $|\mathcal{I}|$ is the total number of instances in the reservoir by the end of the time horizon. This also reflects the impact of having multiple instances in the P-GSMW: each instance get updated for fewer than T times, which makes the job-size variables $\{r_k^I(t)\}$ of each instance converge slower to the optimal compared to the GSMW algorithm in the no-delay setting. To bound the term $\alpha|\mathcal{I}|$, we use the previously obtained result on queueing regret, relate $|\mathcal{I}|$ to the maximum total queue lengths of the network, and show that the utility regret is of order $\tilde{O}(T^{3/4})$. □

4.3.4 Challenges Posed By Feedback Delay. As we have shown in the preceding analysis, the update of job-size variables in the GSMW algorithm (Line 10 of **Algorithm 1**) is composed of one term $V \cdot \hat{\nabla} f_k(\hat{r}_k(t))$ that approximates the gradient and moves the variables towards optimizing utilities, and another term $-Q_{s_k}^k(t)$ that aims at controlling queue lengths. With the presence of feedback delay, GSMW is not applicable as we can only observe the utility values used in the gradient approximation (Line 7) after the jobs get delivered to the destination. Therefore, we may not have the first term available when we perform the updates of GSMW. While we have shown that such difficulty can be overcome by the parallel-instance paradigm, one natural question is **would other simpler adaptation of the GSMW algorithm work?**. We will now look at two other more straightforward modifications of the GSMW algorithm. By arguing at an intuitive level that they are unlikely to work, we further justify the necessity of having the parallel-instance paradigm.

One possible alternative is to use an "episodic approach", i.e., keep the job-size variables unchanged (for an episode of multiple slots) until the utility observations needed become available, and update job-size variables once every episode. Here, as the delay of jobs is essentially proportional to the queue lengths, the length of episodes need to be set to be large enough as the maximum queue length throughout the optimization process. However, this would cause the sizes of the jobs (traffic injected to the network) not be able to adjust timely with respect to the queue lengths (as Line 10 only gets executed once every episode), which will further lead to larger queue lengths (episode length), and thus create a "feedback loop" that makes the algorithm suffer from linear regret.

Another possible method is to use old gradients, e.g., we execute Line 10 every time slot, but use the most recently available gradient estimates. This makes the algorithm adjusts according to queue length every time slot, and thus can maintain the queue length bound of GSMW. However, the feedback delay (queue lengths) is still non-trivial and cannot be bounded by a constant independent of T, which will result in a large bias in the gradient estimates used in the updates and lead to linear regret.

5 APPLICATIONS

In this section, we apply our L-NUM framework to example applications including database query, job scheduling and video streaming.

5.1 Database Query

In this example, we consider a setting where there are K users $\{u_1, \ldots, u_K\}$ querying a central database. At each time t, user u_k issues a query of size r_k to the central database, with r_k representing the processing requirement of the query. The issued queries get buffered in the queue of the database and the database can process c unit of requests in a first-come-first-serve order at each time slot. Each use u_k is associated with an underlying utility function f_k that captures the relationship between the processing requirement

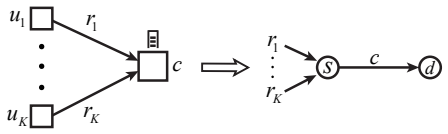

Figure 3: Correspondence between database query and the L-NUM framework.

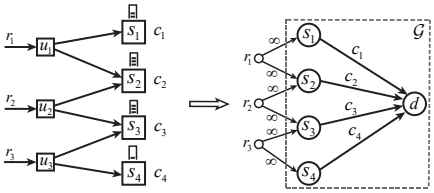

Figure 4: Correspondence between job scheduling and the L-NUM framework.

and utility gained from the query. f_k is Lipschitz continuous and concave, which reflects the diminishing return property of query processing. Over a time horizon of T, the goal is to maximize the total utility of the processed queries.

Applying our framework to the database query example, the network is a simple one with a single state, one source node, one destination node and a link between them (See Figure 3). All the users are mapped to the source node and the database corresponds to the link with the transmission rate of the link at each time slot being equal to the processing capacity c of the database. The network action component of the framework is not needed. The queue at the source node, corresponding to the buffer of the database, buffers the jobs (query requests) of all users. P-GSMW policy adjusts the size of the query according to the gradient estimates and the queue size at the source node, and achieves $\tilde{O}(T^{3/4})$-regret.

5.2 Job Scheduling

Consider a discrete-time system with with a set of job schedulers (dispatchers) $\{u_1, \ldots, u_K\}$ and a set of parallel servers $\{s_1, \ldots, s_M\}$ that form a bipartite graph. We use S_{u_k} to denote the set of servers that dispatcher u_k is connected to. At each time, a class-k job arrives at the dispatcher u_k and the dispatcher sends the job to one of the servers in S_{u_k} for execution. The job dispatcher also determines the resource (e.g. computation, memory) requirement of each job. Each server s_m can provide $c_m(t)$ amount of resources at time t with $c_m(t)$ being a sequence of i.i.d. discrete random variables. Class-k jobs have underlying utility function f_k. A utility of $f_k(r_k)$ is obtained when a class-k job of resource requirement r_k is completed at a server. We seek a scheduling policy that determines the resource requirement and target server of each job. The goal is to maximize the total utility gained from jobs completed over the time horizon T. This example particularly mirrors applications where the jobs are flexible in terms of resource requirement (e.g., model training for machine learning tasks in cloud computing [5, 19]).

We apply the L-NUM framework to the job scheduling application by creating a source node for each job classes, an intermediate node corresponding to each server and a virtual destination node (See Figure 4). The offered transmission rates of the links between server node and the virtual destination is equal to the time-varying capacity $c_m(t)$ of the servers, and the offered transmission rate between source nodes and intermediate nodes are infinity. The job size $r_k(t)$ corresponds to the resource requirement of class k jobs sent at t. Based on this correspondence, the P-GSMW policy achieves $\tilde{O}(T^{3/4})$-regret. Note that the max-weight scheduling component of the P-GSMW is equivalent to the Join-the-Shortest-Queue policy.

5.3 Video Streaming

In this example, we consider a network shared by K users streaming video from K corresponding servers. At each time slot, each server sends a chunk of the video file through the network to its corresponding user. The network operator determine the size of the chunks, which correspond to the rates of the video streams. It also controls the routing and scheduling in the network. User k has a utility function f_k that is unknown to the network operator, and obtains utility of value $f_k(r_k)$ after receiving a video chunk of size r_k. Here, we seek a policy that jointly adapts the video rates, i.e., determines the size of the video chunks and the routing and scheduling of the network such that the total utility obtained from the delivered video chunks is maximized.

It is natural to map the L-NUM framework to the video streaming application. The network shared by the users plays the role of the network $\mathcal{G}$ in the L-NUM framework. Each user represents a traffic class. Each traffic class has the user's corresponding video server as the source node with the user node being the destination. The network states capture the possible time-variability in the network links (e.g. in wireless networks). The network action encapsulates the routing and scheduling actions of the network. The feasible action set $\mathcal{X}$ can captures constraints on network operations such as interference constraints and capacity constraints. Applying the P-GSMW policy, we obtain a joint rate-adaptation and network scheduling/routing policy with $\tilde{O}(T^{3/4})$-regret. The network action component here resembles the back-pressure algorithm.

6 SIMULATIONS

In this section, we evaluate the empirical performance of the P-GSMW policy under the L-NUM framework. We will also compare P-GSMW policy with GSMW algorithm (in an imaginary no-delay setting) to see the impact of feedback delay on the problem.

We instantiate the L-NUM framework on the job scheduling application. The example we construct for the simulation has 50 job schedulers (corresponding to 50 job classes) and 100 parallel servers. The links between job schedulers and servers are randomly generated with each scheduler having expected degree 6 (i.e., connected to 6 servers). The service rate of each server is generated by a uniform random variable with range $[0.5, 1.5]$. We assign an underlying utility function to each class chosen from the four types: $f_k(r) = a_k r$ (linear function), $f_k(r) = a_k\sqrt{r + b_k} - a_k\sqrt{b_k}$ (square root function), $f_k(r) = -a_k r^2 + b_k r$ (quadratic function), $f_k(r) = a_k \log(b_k r + 1)$ (logarithmic function).

Applying the L-NUM framework to the example, we first form the corresponding optimization problem $\mathcal{P}$ and obtain that the optimal value $OPT(\mathcal{P})$ is equal to 84.4. We next run the P-GSMW policy and also the GSMW algorithm (**Algorithm 1**). Note that for the GSMW algorithm, we assume an imaginary no-delay setting where the utility values are immediately observable after decisions.

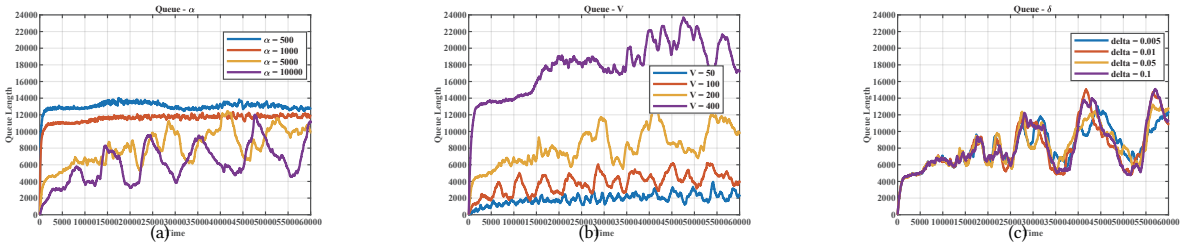

Figure 5: The queue length under the P-GSMW policy with different parameter values.

We first investigate the effects of the parameter values (α, V, δ) on the performance of the policy, then compare P-GSMW and GMSW, and finally study the impact of observation noise.

6.1 Choice of Parameter Values

We vary the values of the parameters (α, V, δ) in the GSMW policy and demonstrate their effects of the policy. The time horizon T is set to 60000. When changing one parameter, the others are held fixed $(\alpha = 5000, V = 200, \delta = 0.005)$. We plot the queue length, defined as the sum of queue length at each server, and the instantaneous utility, defined as $\sum_k f_k(r_k(t))$ as the time evolves. The results on queue length are shown in Figures 5, while the figures on instantaneous utility are much less readible nor informative, and is thus deferred to Appendix B of our technical report [27].

Parameter α: The parameter α essentially controls the step size of the P-GSMW policy with a larger α indicating a smaller step size. We vary α in $\{500, 1000, 5000, 10000\}$. From the results, the average queue length decreases with the increase of α. The queue length of a larger α tends to have larger and more persistent oscillation. Recalling the update of job sizes of the P-GSMW policy (Line 5), such behavior can be attributed to that a larger α leads to a smaller "negative feedback" that the queue length has on job sizes.

Parameter V: The parameter V adjusts the relative weights of the P-GSMW policy on utility maximization and queue stability, with a larger V indicating that the policy tries to increase the job sizes (and thus the instantaneous utility) more aggressively. We vary V in $\{50, 100, 200, 400\}$. Such behavior is clearly reflected in Figure 5(b) as a larger V leads to a larger steady-state queue size, but the difference is more obscure in the plot of instantaneous utility (figure omitted due to space constraint). We further calculate the time-average instantaneous utility of the P-GSMW policy under different values of V. Corresponding to $V = 50, 100, 200, 400$, the time-average instantaneous utility are 84.8, 85.3, 86.1, 87.1, respectively. Note that the instantaneous utility can be larger than $OPT(\mathcal{P})$ since the virtual job size variables may not satisfy the capacity constraint of $\mathcal{P}$. The result further supports that a larger V leads to more aggressive increase in the job sizes.

Parameter δ : The parameter δ controls the approximation error of our estimate gradients with respect to the true gradients. We vary δ in $\{0.005, 0.01, 0.05, 0.1\}$. Due to that the underlying utility functions in our example do not have large curvature, the value of δ does not have significant effect on the policy.

6.2 P-GSMW vs. GSMW

We compare the behaviors of total queue length and instantaneous utility under P-GSMW and GSMW with the same parameter values of $(\alpha = 5000, V = 200, \delta = 0.005)$. Note that P-GSMW is run in our original setting (with queueing-style feedback delay) while GSMW is run in an imaginary no-delay setting. It can be seen that, as in terms of job-size variables, P-GSMW switches between different instances of GSMW algorithms, both the queue length trajectory and the instantaneous utility trajectory under P-GSMW exhibits larger oscillation compared to those of GSMW.

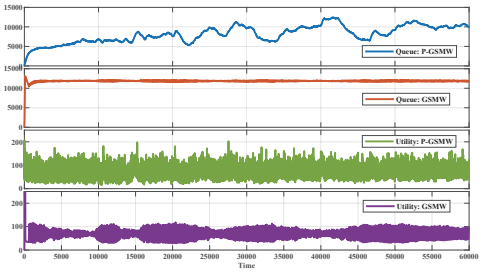

Figure 6: Queue length and instantaneous utility behavior under P-GSMW and GSMW.

Furthermore, varying the time horizon T in $\{10000, 20000, \ldots, 100000\}$ and setting $\alpha = 50\sqrt{T}, V = T^{1/4}, \delta = 1/\sqrt{T}$, we compare how the regret of P-GSMW and GSMW scales with the time horizon. Since it is computationally infeasible to compute the optimal strategy, we use T times $OPT(\mathcal{P})$ as an upper bound of the expected utility achieved by the optimal strategy (see Theorem 1) and bound the regret by $T \cdot OPT(\mathcal{P})$ minus the utility achieved by the policies. We can see from Figure 7 that the regret of GSMW is lower by that of P-GSMW, which suggests that the feedback delay hurt the performance of the policy.

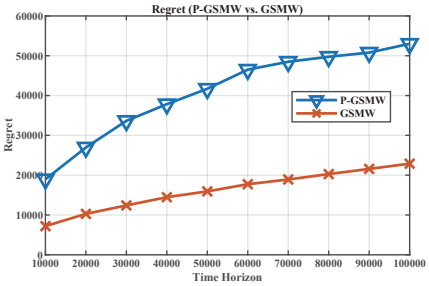

Figure 7: Regrets of P-GSMW and GSMW.

6.3 Observation Noise

We explore the situation where the utility observations are corrupted with noise and study the robustness of P-GSMW against such noise. We change the noise level from 0 (no noise) to 0.2 (each

observation is corrupted with noise that is uniformly distributed in $[-0.2, 0.2]$). Varying the time horizon in $\{10000, 20000, \ldots, 100000\}$ and setting $\alpha = 50\sqrt{T}, V = T^{1/4}, \delta = 1/\sqrt{T}$, we evaluate the scaling of regret under different noise levels. The results are plotted in Figures 8.

From Figures 8, we see that the regrets of P-GSMW sublinear growth with time horizon even under a noise level of 0.2. Generally, the regret increases with noise level, but the difference is not significant for noise under 0.05.[3] The degradation of regret performance with noise can be attributed to that the variance of the gradient estimate. Recall that the gradient estimates of the P-GSMW policy at a time t for class k is equal to $\hat{\nabla} f_k(\hat{r}_k^I(t)) := \frac{f_k(\hat{r}_k^I(t)+\delta) - f_k(\hat{r}_k^I(t)-\delta)}{2\delta}$. If the two observations are corrupted by random noise ϵ_1, ϵ_2 respectively, then we have $\mathbb{E}[|\hat{\nabla} f_k(\hat{r}_k^I(t))|] \simeq \mathbb{E}[|\frac{f_k(\hat{r}_k^I(t)+\delta) - f_k(\hat{r}_k^I(t)-\delta)}{2\delta}| + \frac{|\epsilon_1 - \epsilon_2|}{2\delta}]$. Due to the Lipschitz-continuity of f_k, the first term is of order $O(1)$ but the second is of $O(1/\delta)$. Thus, the second term dominates the magnitude of the gradient estimate and it increases with the magnitude of ϵ (i.e., the noise level). A gradient estimate of larger magnitude may lead to less stable updates, larger queue lengths, longer feedback delay, and ultimately, larger regret.

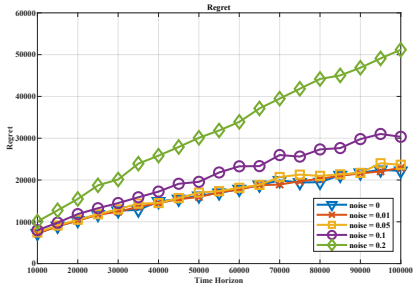

Figure 8: Regret of P-GSMW under different noise levels.

7 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we proposed a new NUM framework, Learning-NUM, where the utility functions are only accessible through zeroth-order feedback and the feedback experiences queueing-style delay. We upper bounded the expected utility achieved by any dynamic policy by the solution to a static optimization problem and designed an online scheduling policy (P-GSMW) that achieves sub-linear regret.

Our scheduling policy achieves a regret of order $\tilde{O}(T^{3/4})$. This is worse than the existing lower bound of $\Omega(\sqrt{T})$, which was shown in the no-delay case. Hence, an important future direction is to determine whether the queueing-style delay of L-NUM fundamental increases the difficulty of the problem, i.e., a lower bound better than $\Omega(\sqrt{T})$ can be established, or that algorithm for L-NUM that has regret better than $\tilde{O}(T^{3/4})$ exists. Finally, we have not theoretically studied the performance of P-GSMW policy under observation noise. Although it is expected that P-GSMW would have regret worse than $\tilde{O}(T^{3/4})$, how to minimize the adverse impact of the noise on the policy, and are there other methods that are more robust to noise are both questions of future interests.

[3]To put this into perspective, there are 50 job classes and $OPT(\mathcal{P})$ is 84.4. The magnitude of the noise is about $0.05 \times 50/84.4 \simeq 3\%$ of the time-average utility.

8 ACKNOWLEDGMENTS

This work was funded by NSF grant CNS-1524317 and by Office of Naval Research (ONR) grant award N00014-20-1-2119.

REFERENCES

[1] F. P. Kelly, K. Aman, Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability." in *Journal of the Operational Research Society*, Vol. 49, No. 3, pp: 237-252, 1998.
[2] S. H. Low and D. E. Lapsley, "Optimization flow control. I. Basic algorithm and convergence." in *IEEE/ACM Transactions on Networking*, Vol. 7, No .6, pp: 861-874, 1999.
[3] D. P. Palomar and M. Chiang, "Alternative distributed algorithms for network utility maximization: Framework and applications." in *IEEE Transactions on Automatic Control*, Vol. 52, No.12, pp: 2254-2269, 2007.
[4] M. J. Neely, E. Modiano, and C. E. Rohrs, "Dynamic power allocation and routing for time varying wireless networks." in *IEEE INFOCOM*, 2003.
[5] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters." in *IEEE INFOCOM*, 2012.
[6] J. Ghaderi, S. Shakkottai, and R. Srikant, "Scheduling storms and streams in the cloud." in *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, Vol. 1, No. 4, pp: 1-28, 2016.
[7] X. Lin and N. B. Shroff, "Utility maximization for communication networks with multipath routing." in *IEEE Transactions on Automatic Control*, Vol. 51, No. 5, pp: 766-781, 2006.
[8] E. Wei, A. Ozdaglar, and A. Jadbabaie, "A distributed newton method for network utility maximization.", in *IEEE Conference on Decision and Control (CDC)*, 2010.
[9] M. J. Neely, E. Modiano, and C. P. Li, "Fairness and optimal stochastic control for heterogeneous networks." in *IEEE/ACM Transactions On Networking*, Vol. 16, No.2, pp: 396-409, 2008.
[10] L. Huang and M. J. Neely, "Utility optimal scheduling in energy-harvesting networks." in *IEEE/ACM Transactions on Networking*, Vol. 21, No.4, pp: 1117-1130, 2012.
[11] Q. Liang and E. Modiano, "Network utility maximization in adversarial environments." in *IEEE INFOCOM*, 2018.
[12] M. Hajiesmaili, A. Khonsari, A. Sehati, and M. S. Talebi, "Content-aware rate allocation for efficient video streaming via dynamic network utility maximization." in *Journal of Network and Computer Applications*, Vol. 35, No. 6, pp: 2016-2027, 2012.
[13] D. Bethanabhotla, C. Giuseppe Caire, and M. J. Neely, "Adaptive video streaming for wireless networks with multiple users and helpers." in *IEEE Transactions on Communications*, Vol. 63, No.1, pp: 268-285, 2014.
[14] Y. Zheng, J. Bo, N. Shroff, and P. Sinha, "Forget the deadline: Scheduling interactive applications in data centers." in *IEEE International Conference on Cloud Computing*, pp: 293-300, 2015.
[15] A. D. Flaxman, A. T. Kalai, and H. B. McMahan, "Online convex optimization in the bandit setting: gradient descent without a gradient", in *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pp: 385-394, 2005.
[16] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks." in *IEEE Conference on Decision and Control*, 1990.
[17] H. Yu, M. Neely, and X. Wei, "Online convex optimization with stochastic constraints." in *Advances in Neural Information Processing Systems*, 2017.
[18] H. Pang, M. J. Carey, and M. Livny, "Multiclass query scheduling in real-time database systems." in *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 4, pp: 533-551, 1995.
[19] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning." in *Proceedings of the Symposium on Cloud Computing*, pp: 390-404, 2017.
[20] S. Shalev-Shwartz, "Online learning and online convex optimization." in *Foundations and Trends in Machine Learning*, Vol. 4, No. 2, pp: 107-194, 2012.
[21] A. Agarwal, D. P. Foster, D. J. Hsu, S. M. Kakade, and A. Rakhlin, "Stochastic convex optimization with bandit feedback." in *Advances in Neural Information Processing Systems*, pp: 1035-1043, 2011.
[22] S. Yang and M. Mohri, "Optimistic bandit convex optimization." in *Advances in Neural Information Processing Systems*, pp: 2297-2305, 2016.
[23] O. Shamir, "On the complexity of bandit and derivative-free stochastic convex optimization." in *Conference on Learning Theory*, 2013.
[24] T. Chen, G. B. Giannakis, "Bandit convex optimization for scalable and dynamic IoT management.", in *IEEE Internet of Things Journal*, Vol. 6, No. 1, pp: 1276-1286.
[25] P. Joulani, A. Gyorgy, and C. Szepesvári. "Online learning under delayed feedback." in *International Conference on Machine Learning*, 2013.
[26] P. Joulani, A. György, and C. Szepesvári. "Delay-Tolerant Online Convex Optimization: Unified Analysis and Adaptive-Gradient Algorithms." in *AAAI*, 2016.
[27] X. Fu and E. Modiano, "Learning-NUM: Network Utility Maximization with Unknown Utility Functions and Queueing Delay", Technical Report, http://arxiv.org/abs/2012.09222.