A Backpropagation Approach for Distributed Resource Allocation

Alexandre Reiffers-Masson, Nahum Shimkin, Daniel Sadoc Menasche, Eitan Altman

A Backpropagation Approach for Distributed Resource Allocation

ALEXANDRE REIFFERS-MASSON, NAHUM SHIMKIN, DANIEL SADOC MENASCHE AND EITAN ALTMAN

Network resource allocation through Network Utility Maximization (NUM) is one of the fundamental problems in the realm of networked systems. In the NUM framework, the network comprises a set of nodes each of which is associated with a utility function, and the goal is to distribute resources across nodes so as to maximize the sum of the nodes utilities. In this paper, we propose a novel backpropagation approach for distributed resource allocation. The internal flow of resources among nodes is governed by the network dynamics, assumed to be captured through a directed acyclic graph (DAG). Control is exercised as an external injection of limited resources at some nodes, where the goal is to determine the optimal amount of resources to be injected at nodes, under the NUM framework. To that aim, we present a novel forward-backward algorithm, inspired by neural network training, wherein flows of resources are transferred during the forward step, and gradients are backpropagated at the backward step. Based on such gradients, the controls are adjusted, considering two variations of the algorithm under synchronous and asynchronous settings. The proposed algorithms are distributed, in the sense that information in transferred only between neighboring nodes in the network. In addition, they are suitable for continued operation, so that the optimum resource allocation is tracked as conditions gradually change. We formally establish convergence of the proposed algorithms, and numerically compare the speed of convergence under the asynchronous setting against its synchronous counterpart. Together, our results advance the state-of-the-art in the realm of NUM under nonlinear constraints, indicating how to leverage a backpropagation approach for that matter.

ACM Reference format:
Alexandre Reiffers-Masson, Nahum Shimkin, Daniel Sadoc Menasche and Eitan Altman. 2016. A Backpropagation Approach for Distributed
Resource Allocation. 1, 1, Article 1 (January 2016), 12 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Network resource allocation is one of the fundamental problems in the realm of networked systems. In essence, the problem consists in determining how to distribute limited resources across nodes in a network, so as to maximize a given utility function. In particular, under the Network Utility Maximization (NUM) framework [9, 15], each node is associated to a utility, and the goal is to maximize the sum of utilities. The generality of the framework allows it to capture a broad range of scenarios, ranging from physical networks, such as supply-chains, up to virtual networks, such as virtual markets.

In this paper, we propose a novel backpropagation approach for distributed resource allocation. The internal flow of resources among nodes is governed by network dynamics, assumed to be captured through a directed acyclic graph (DAG). Control is exercised as an external injection of

DOI: 10.1145/nnnnnnn.nnnnnnn

limited resources at some nodes. The goal is to determine the optimal amount of resources to be injected at each node.

The total flow of resources received by a node yields a corresponding utility. Similarly, the external injection of resources at any node yields a cost. Utilities (resp. costs) are assumed to be concave (resp. convex) increasing, and are allowed to be nonlinear. The function which dictates the amount of flow received by a node that is forwarded to its neighbors, referred to as the interaction function, is also assumed to be concave and increasing, and possibly nonlinear. Each node is associated to its own interaction function. As flow dynamics are constrained by nonlinear interaction functions, the NUM problems considered in this work are subject to nonlinear constraints.

We pose the following questions:

- How to allocate limited resources to networked nodes so as to solve a NUM problem under nonlinear constraints?
- How do synchronous solutions compare against their asynchronous counterparts?

For the optimal resource allocation, we design a novel forward-backward algorithm, inspired by neural network training, wherein flows of resources are transferred during the forward step, and gradients are backpropagated at the backward step. Based on such gradients, the controls are adjusted. We consider two variations of the algorithm under synchronous and asynchronous settings. Backpropagation is one of the pillars of artificial intelligence, and one of our insights consists in leveraging backpropagation for resource allocation purposes, showing its applicability in addressing NUM problems.

The proposed algorithms are distributed, in the sense that nodes only need to exchange local information with their neighbors. In addition, our algorithms are suitable for continued operation. Indeed, they allow for the optimum resource allocation to be tracked as conditions gradually change.

We formally establish convergence of the proposed algorithms, and numerically compare the speed of convergence of the asynchronous setting against its synchronous counterpart. Together, our results advance the state-of-the-art in the realm of NUM under nonlinear constraints, indicating how to leverage a backpropagation approach for that matter.

Summary of contributions. The main contributions of this paper are given below:

- **Nonlinear resource allocation problem formulation:** We formulate a NUM resource allocation problem under nonlinear constraints. The problem consists of maximizing the sum of utilities of nodes in a network where the utilities are a function of the received flows. We assume that the flow constraints are concave but possibly nonlinear, and show the convergence of the proposed backpropagation approach towards a NUM solution.
- **Design of efficient algorithms:** We provide two efficient algorithms which converge to the solution of the first-order optimality conditions of our optimization problem. The first algorithm is based on the observation that the first-order condition can be solved using a forward-backward approach. The second algorithm is an extension of the first one, by allowing asynchronous communications. Finally, as mentioned above, both algorithms are distributed in the sense that nodes only need to exchange local information with their neighbors. We illustrate the performance of our algorithms through a numerical study.

Outline. In the following section we report related work. Then, we introduce the proposed model (Section 3), followed by synchronous and asynchronous solutions in Section 4. Numerical evaluation is reported in Section 5 and Section 6 concludes.

2 RELATED WORK

Next, we report related work on main themes of the paper, namely classical and state-of-the-art network optimization methods (Sections 2.1 and 2.2) and strategies to handle nonlinear constraints under NUM (Section 2.3).

2.1 Primal-dual and saddle point algorithms

Our optimization problem belongs to the class of nonlinear network optimization problems [4]. Most of the recent research on the design of efficient algorithms for convex objective functions accounts for linear flow constraints. Next, we briefly overview some of those related efforts.

A variety of approaches have been considered under linear flow constraints, including distributed interior point methods [8] or distributed primal-dual algorithms [10, 17]. To the best of our knowledge, one of the first works to design a distributed algorithm for NUM with convex flow constraints is [11]. The authors use Lagrange multipliers and propose a distributed price allocation algorithm which converges to the optimal solution of the network problem. Nonetheless, the flow constraints in [11] are different from ours, motivating a new solution. Indeed, the specifics of our constraints allow us to design novel synchronous and asynchronous algorithms.

Similarly, in [2, 3] the authors also suggest a decentralized asynchronous iterative algorithm to solve a NUM instance with nonlinear stochastic constraints. The authors leverage a Lagrange relaxation of the problem to derive an asynchronous saddle point iterative algorithm, which converges to the optimal solution. The algorithms proposed in the sequel also belong to the class of saddle point iterative algorithm. Nonetheless, the considered constraints do not provide an explicit characterization of the interaction between the actions of the different agents. Therefore, in our setting the computation of the gradients motivates the proposed backpropagation approach.

2.2 The push-pull method for network optimization and distributed learning

A push–pull gradient method has been recently considered to solve the optimal flow allocation problem in a distributed fashion [12, 18]. Under the push-pull method, each node keeps in memory the current estimate of its optimal decision variable as well as an estimate of the gradient of the agents objective function. Then, each agent transfers information about the gradients to its neighbors, i.e., such information is pushed downstream, whereas information about decisions is pulled from the neighbors, i.e., it is pulled upstream.

Push-pull methods have been considered for distributed machine learning applications [18], with the goal of collectively learning average values of elements distributed across the network, through consensus. In this work we are also inspired by push-pull gradient methods, but consider a different goal, namely solving a resource allocation problem in a distributed setup. Indeed, we leverage the intuitive resemblance between push–pull gradient methods and backpropagation to propose a novel resource allocation mechanism, accounting for nonlinear constraints.

2.3 Handling chance-constraints and general nonlinear constraints under NUM

Chance-constraints are another sort of nonlinear constraints that have also been accounted for by the NUM framework [14]. Consider, for instance, chance-constraints indicating that the tail probability of the delay distribution should be bounded by a given threshold. Under chance-constraints, the feasible set may be non-convex, motivating the restriction to a convex subset where a near-optimal solution exists [14]. The use of surrogate linear constraints have been considered for that matter, leveraging the Markov [16], Chebyshev or Bernstein approximations [19].

Distributed algorithms for convex optimization problems with nonlinear convex constraints have also been considered in [13]. One of the steps in the algorithm proposed in [13] consists of

Table 1. Table of notation

variable	description
x_i	current allocation of (external and internal) resources to node i
u_i	amount of external resource allocated to node i
	($x_i - u_i$ is the amount of internal resources allocated to node i)
$y_i = f_i(x_i)$	amount of resources from node i redistributed to its neighbors
w_{ij}	fraction of resources from node i shared with node j (flow from i to j)

a projection step. Such a step is non-trivial in the resource allocation problem considered in this work. Therefore, we leverage the specifics of our optimization problem which allow us to design alternative simpler algorithms.

3 MODEL AND OPTIMALITY CONDITIONS

Next, we introduce the proposed model (Section 3.1) followed by an analysis of its optimality conditions (Section 3.2) and an illustrative example (Section 3.3).

3.1 Model

We consider a set of I nodes, also referred to as agents, $\mathcal{I} := \{1, \dots, I\}$. Agents are connected through a *directed acyclic graph* (DAG), where relationships are described by an adjacency matrix $\mathbf{W}$. The ij-entry of $\mathbf{W}$, denoted by $w_{ij} \in [0, 1]$, measures the fraction of resources/flow an agent i is sharing with j.

Nodes are divided into internal and leaf nodes. For any node i we have $w_{ii} = 0$. In addition,

$$\sum_{j=1}^{I} w_{ij} = \begin{cases} 1, & \text{if } i \text{ is an internal node,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Let $x_i \in \mathbb{R}_+$ be the amount of resources received by agent i and let $y_i \in \mathbb{R}_+$ be the amount of resources agent i is redistributing to its neighbors. An agent can receive resources from its neighbors and from an external source. Let $u_i \in \mathbb{R}_+$ denote the amount of external resources received by agent i. For each agent i, x_i and y_i are related as follows:

$$x_i = u_i + \sum_{j=1}^{I} w_{ji} y_j, \; y_i = f_i(x_i), \tag{2}$$

where $f_i(\cdot)$ is a continuous increasing concave differential function. Function f_i is referred to as the *interaction function*, and captures the amount of resources agent i is redistributing to its neighbors. Note that y_i and x_i are concave in $\mathbf{u} := [u_1, \dots, u_I]$ as long as for every i, f_i is concave increasing. Our goal is to design a distributed algorithm which finds the optimal amount of external resources $\mathbf{u}$ which maximizes a given utility function $U(\mathbf{x}) := \sum_{i=1}^{I} U_i(x_i)$ minus a cost $C(\mathbf{u}) := \sum_{i=1}^{I} C_i(u_i)$ associated to the allocation of those resources. For every $i \in \mathcal{I}$, we assume that $U_i(\cdot)$ and $-C_i(\cdot)$ are continuous concave and differentiable functions.

In summary, our algorithm should converge to a local optimum of the following optimization problem:

$$\begin{aligned} &\max_{\mathbf{u}\in[\underline{u},\overline{u}]^I} \quad \sum_{i=1}^{I} U_i(x_i) - \sum_{i=1}^{I} C_i(u_i) := U(\mathbf{x}) - C(\mathbf{u}), \\ &\text{subject to } \mathbf{x} \text{ solution of:} \\ &\qquad x_i = u_i + \sum_{j=1}^{I} f_j\left(x_j\right) w_{ji}, \ \forall i \in \{1,\dots,I\}, \end{aligned} \tag{3}$$

where $\underline{u}$ and $\overline{u}$ are the lower and upper bounds on the amount of external resources that can be allocated to a node.

3.2 Optimality conditions

Next, we express the Lagrangian of our optimization problem and the corresponding Karush Kuhn Tucker (KKT) conditions (see [6]). The Lagrangian is given by

$$\mathcal{L}(x,u,\lambda) = \sum_{i=1}^{I} U_i(x_i) - \sum_{i=1}^{I} C(u_i) + \sum_{i=1}^{I} \lambda_i \left(u_i + \sum_{j=1}^{I} f_j(x_j) w_{ji} - x_i \right), \tag{4}$$

where $\lambda = [\lambda_i]_{1\le i\le I}$ are the Lagrange multipliers. Note that we do not take into account the boundary constraints $[\underline{u},\overline{u}]^I$ in the Lagrangian. Instead, we incorporate them directly in the gradient ascent update, to be described in the sequel.

Differentiating the above equation with respect to x_i we obtain the corresponding first order condition,

$$0 = U_i'(x_i) - \lambda_i + f_i'(x_i) \sum_{j=1}^{I} w_{ij}\lambda_j, \tag{5}$$

where $f_i'(x_i)$ is the derivative of f_i with respect to x_i. Differentiating $\mathcal{L}(x,u,\lambda)$ with respect to u_i we obtain

$$0 = -C_i'(u_i) + \lambda_i. \tag{6}$$

If $(\mathbf{u}^*,\mathbf{x}^*)$ is a strict local maximizer of (3) and if $f_i(\cdot)$ is concave and increasing, $U_i(\cdot)$ is concave and $C_i(\cdot)$ is convex for every $i \in \{1,\dots,I\}$, then $(\mathbf{u}^*,\mathbf{x}^*,\lambda^*)$ is a solution of the above local optimality conditions (equations (5)-(6)).

3.3 Network models in Economy: network shocks *versus* network control

Next, we illustrate the applicability of the proposed framework in the realm of a virtual market. To that aim, we consider the general framework introduced in [1] to analyze the role of network interactions in the macroeconomic performance of a given Economy. Consider an Economy comprised of I agents, where $x_i \in \mathbb{R}$ is the state of agent $i \in \{1,\cdots,I\}$. The joint state of all agents is denoted by $\mathbf{x} = [x_i]_{1\le i\le I}$. In [1] the authors seek for a solution of the following fixed point problem

$$x_i = \tilde{f}\left(\sum_{j=1}^{I} w_{ij}x_j + u_i \right) \tag{7}$$

for all $i \in \{1,\dots,I\}$, where the interaction function $\tilde{f}$ is a continuous and increasing function. The weights w_{ij} capture the intensity of the interaction between i and j, with $w_{ij} \in [0,1]$ and $\sum_j w_{ij} = 1$. Finally, the utility function in [1] is assumed to be of the form $U(\mathbf{x}) = g(\sum_{i=1}^{I} h(x_i))$, and is referred to as the macro state of the Economy.

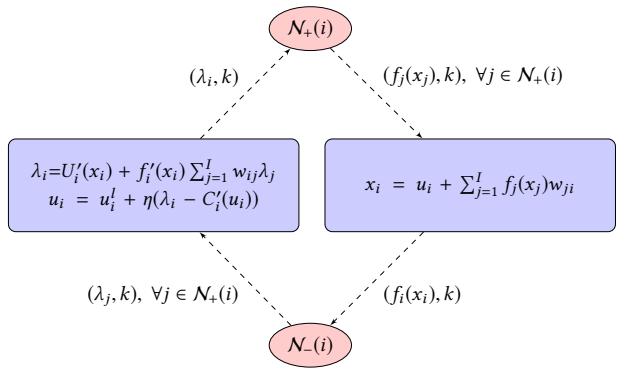


Fig. 1. Block diagram of the synchronous resource allocation distributed algorithm, at the level of node i. Upstream neighbors ($\mathcal{N}_+(i)$) transfer flows at the forward step, and downstream neighbors ($\mathcal{N}_-(i)$) backpropagate gradients at the backward step. The current time slot, k, is included in all messages.

Clearly, the above economic model is a special instance of the problem formulation considered in our work. Nonetheless, in [1] the term u_i is assumed to be an agent-level shock, capturing stochastic disturbances to an agent state. The authors assume that these external shocks are independently and identically distributed. In what follows, in contrast, we consider a controllable external input of resources, that is our main object of study.

4 DISTRIBUTED ALGORITHMS

In this section, we design two algorithms and prove their convergence towards an optimal solution of the optimization problem introduced in the previous section. Both algorithms rely on distributed computation of the Lagrange multipliers, following a gradient ascent approach inspired by the concept of backpropagation. The first algorithm is synchronous, and the second extends it by introducing multiple timescales that allow nodes to perform asynchronous computation.

4.1 Distributed gradient algorithm for convex flow constraints

For every node $i \in \mathcal{I}$, we define the sets of upstream and downstream neighbors, $\mathcal{N}_+(i) = \{j \in \mathcal{I} \mid w_{ji} > 0\}$ and $\mathcal{N}_-(i) = \{j \in \mathcal{I} \mid w_{ij} > 0\}$, respectively. The main idea behind the algorithm design consists in leveraging the fact that to compute λ_i^* (resp. x_i) node i only needs to retrieve $\lambda_j^*,\ \forall j \in \mathcal{N}_-(i)$ (resp. $f_j(x_j),\ \forall j \in \mathcal{N}_+(i)$). This idea is similar in spirit to backpropagation in neural networks, where gradients are computed one layer at a time, iterating backward from the last layer. The latter corresponds to leaf nodes in our networks, i.e., nodes i such that $\mathcal{N}_-(i) = \emptyset$. Indeed, the proposed algorithm extends backpropagation to a DAG topology, generalizing the typical multilayer topology considered for machine learning applications. The proposed algorithm is described below:

Backpropagation Algorithm for Distributed Resource Allocation
Initialization: Set the step-size $\eta \in (0, 1)$ and initialize $u_i(0) = u_i^0$ for all $i \in \mathcal{I}$.
Updates: At each iteration $k = 1, 2, \ldots$:

(1) *Forward step: computation of $x_i(k)$.* Each node $i \in \mathcal{I}$ computes x_i through the following update rule:

$$x_i(k) = u_i(k) + \sum_{j \in \mathcal{N}_+(i)} f_j(x_j(k)) w_{ji}. \tag{8}$$

(2) *Backward step: computation of* $\lambda_i(k)$. Each node $i \in \{1, \ldots, I\}$ computes $\lambda_i(k)$ by using $x_i(k)$ and the following update rule:

$$\lambda_i(k) = U_i'(x_i(k)) + f_i'(x_i(k)) \sum_{j \in \mathcal{N}_-(i)} w_{ij}\lambda_j(k).$$

(3) *Gradient ascent step.* Each node $i \in \mathcal{I}$ computes $u_i(k+1)$ as follows:

$$u_i(k+1) = \left[u_i(k) + \eta(-C_i'(u_i(k)) + \lambda_i(k)) \right]_{\underline{u}}^{\overline{u}}, \tag{9}$$

where $[\cdot]_{\underline{u}}^{\overline{u}}$ is the projection operator, $[x]_{\underline{u}}^{\overline{u}} := \min\{\max\{x, \underline{u}\}, \overline{u}\}$.

It follows from (5)-(6) that the proposed algorithm converges to a local optimum.

Note that the above algorithm is distributed in the sense that a node only needs to receive information from its direct neighbors $\mathcal{N}_-(i)$ and $\mathcal{N}_+(i)$ to update x_i, λ_i and u_i. Moreover, the algorithm can be implemented in an event driven framework. However, its main drawback concerns the speed of convergence, which depends on the computation power of the slowest nodes. A node i has to wait all nodes $j \in \mathcal{N}_+(i)$ to compute $x_i(k)$. Similarly a node i has to wait all nodes $j \in \mathcal{N}_-(i)$ to compute $\lambda_j(k)$. Such observation motivates an asynchronous solution as described in the sequel.

4.2 Distributed asynchronous version

Next, we design an asynchronous algorithm for the considered NUM problem. To that aim, we introduce three time scales. Let $Y_k \subset \{1, \ldots, I\}$ be the random subset of activated nodes at time k. Let $\nu(i,k)$ be the number of updates at node i by time k. Then, $\nu(i,k) := \sum_{k'=0}^{k} \mathbb{I}\{i \in Y_{k'}\}$, where $\mathbb{I}\{c\}$ is an indicator variable, which equals 1 if condition c is satisfied, and 0 otherwise. The three step-sizes $\{a(k)\}, \{b(k)\}, \{b(k)\} \in (0,1)$ corresponding to three time scales are such that:

(1) $\sum_{k=1}^{\infty} a(k) = \sum_{k=1}^{\infty} b(k) = \sum_{k=1}^{\infty} c(k) = \infty$,
(2) $\sum_{k=1}^{\infty} a^2(k) + b^2(k) + c^2(k) < \infty$,
(3) $\lim_{k\to\infty} \frac{b(k)}{a(k)} = 0$ and $\lim_{k\to\infty} \frac{c(k)}{b(k)} = 0$.

It what follows, we describe the update at a given node i.

Distributed Asynchronous Gradient Algorithm (Local iteration of node i)

Initialization: Set $x_i(0) = x_i^0$, $\lambda_i(0) = \lambda_i^0$ and $u_i(0) = u_i^0$.

(1) *Pull step:* If node $i \in Y_k$, node i

- requests the outputs from its neighbors $\mathcal{N}_+(i)$ and $\mathcal{N}_-(i)$,
- receives $f_j(x_j(k - \tau_{ij}(k)))$ from all $j \in \mathcal{N}_+(i)$, and $\lambda_j(k - \tau_{ij}(k))$ from all $i \in \mathcal{N}_-(i)$.

(2) *Update step:* Node i performs the following updates:

$$x_i(k+1) = x_i(k) + a(\nu(i,k))\mathbb{I}\{i \in Y_k\}\left(u_i(k) + \sum_{j\in\mathcal{N}_+(i)} f_j(x_j(k-\tau_{ij}(k)))w_{ji} - x_i(k)\right), \tag{10}$$

$$\lambda_i(k+1) = \lambda_i(k) + b(\nu(i,k))\mathbb{I}\{i \in Y_k\}\left(U_i'(x_i(k)) - \lambda_i(k) + f_i'(x_i(k))\sum_{j\in\mathcal{N}_-(i)} w_{ij}\lambda_j(k-\tau_{ij}(k))\right), \tag{11}$$

$$u_i(k+1) = \left[u_i(k) + c(\nu(i,k))\mathbb{I}\{i \in Y_k\}\left(\lambda_i(k) - C_i'(u_i(k))\right)\right]_{\underline{u}}^{\overline{u}}. \tag{12}$$

The proposed distributed asynchronous algorithm is a three time scale stochastic approximation.

(1) *Fast Time Scale:* given by (10), it eventually converges to (2). The elements updated through slower time scales, given by (11) and (12), are taken as static under this standpoint;
(2) *Middle Time Scale:* given by (11), it tracks the Lagrange multiplier λ_i for all $i \in \mathcal{I}$, and eventually converges to (5). It relies on a solution of (10) for a fixed u;
(3) *Slow Time Scale:* the gradient ascent given by (9) to track (6) is performed on the slow time scale, relying on estimates of λ_i for all $i \in \mathcal{I}$.

4.3 Convergence analysis of the asynchronous algorithm: the ODE approach

Next, we establish the convergence of the asynchronous algorithm towards a local maximum. To that aim, we rely on the ordinary differential equation (ODE) approach. We begin by establishing the ODEs corresponding to the stochastic approximation. Then, we show that the ODEs converge to a solution satisfying the KKT conditions presented in Section 3.2. Finally, we indicate that mild assumptions required for the stochastic approximation to track the ODEs, established in [5], are satisfied, which concludes the argument.

4.3.1 Establishing the ODEs. We consider the following singularly perturbed ordinary differential equations,

$$\dot{x}_i = u_i + \sum_{j\in\mathcal{N}_+(i)} w_{ji}f_j(x_j) - x_i, \tag{13}$$

$$\dot{\lambda}_i = \epsilon_1\left(U_i'(x_i) - \lambda_i + f_i'(x_i)\sum_{j\in\mathcal{N}_-(i)} w_{ij}\lambda_j\right), \tag{14}$$

$$\dot{u}_i = \epsilon_2(C_i'(u_i) - \lambda_i - v_i(\mathbf{u})), \tag{15}$$

$\forall i \in \{1,\dots,I\}$, with $0 < \epsilon_1 \downarrow 0$, $0 < \epsilon_2 \downarrow 0$, $\frac{\epsilon_2}{\epsilon_1} \downarrow 0$ and where $\mathbf{v}(\mathbf{u}) = [v_i(\mathbf{u})]_{1\le i\le I}$ is the minimum adjustment needed to keep $\mathbf{u}$ in $[\underline{u},\overline{u}]^I$, noting that (15) is a projected dynamical system (see Chapter 4 in [7]).

Next, we establish convergence of the above ODEs under the fast, medium and slow time scales, respectively.

4.3.2 Convergence of ODEs. The fast timescale updates correspond to the following ODEs,

$$\dot{x}_i = u_i + \sum_{j\in\mathcal{N}_+(i)} w_{ji}f_j(x_j) - x_i, \quad \dot{\lambda}_i = 0, \quad \dot{u}_i = 0, \quad \forall i \in \{1,\dots,I\}. \tag{16}$$

As we consider a DAG topology, for any given $\mathbf{u}$ the solution of (16) has a unique globally stable equilibrium, denoted by $x_i^*(\mathbf{u})$, which is the solution of:

$$0 = u_i + \sum_{j \in \mathcal{N}_+(i)} w_{ji} f_j(x_j^*(\mathbf{u})) - x_i^*(\mathbf{u}),\ \forall i \in \mathcal{I}.$$

Updates at the middle time scale correspond to the following ODEs,

$$0 = u_i + \sum_{j \in \mathcal{N}_+(i)} w_{ji} f_j(x_j^*(\mathbf{u})) - x_i^*(\mathbf{u}), \tag{17}$$

$$\dot{\lambda}_i = U_i'(x_i^*(\mathbf{u})) - \lambda_i + f_i'(x_i^*(\mathbf{u})) \sum_{j \in \mathcal{N}_-(i)} w_{ij} \lambda_j, \tag{18}$$

$$\dot{u}_i = 0, \tag{19}$$

$\forall i \in \{1, \ldots, I\}$. The convergence of (18) also follows from the assumption that the considered network has a DAG topology, and the fact that $x_i(t)$ has already converged to $x_i^*(\mathbf{u})$, for any given $\mathbf{u}$. Then, $\lambda_i(t)$ converges to $\lambda_i(x_i^*(\mathbf{u}))$, which is a solution of

$$0 = U_i'(x_i^*(\mathbf{u})) - \lambda_i + f_i'(x_i^*(\mathbf{u})) \sum_{j \in \mathcal{N}_-(i)} w_{ij} \lambda_j(x_j^*(\mathbf{u})).$$

The convergence of the slow time scale follows from the fact that $\mathcal{L}(x^*(\mathbf{u}), \mathbf{u}, \lambda(x^*(\mathbf{u})))$ is a Lyapunov function corresponding to the gradient ascent (15).

4.3.3 Stochastic approximation tracks ODEs. Next, we indicate the mild assumptions over Y_n and $\tau_i(\cdot)$ that ensure that if the above ODEs converge to a globally stable point then the corresponding stochastic approximation scheme also converges to the same point. The assumptions can be found in Chapter 7 of [5], and are summarized as follows.

Assumption over $\{Y_n\}$. Assume that $\{Y_n\}$ is an irreducible Markov chain, independent of $\{\mathbf{x}(k)\}_{k=1}^{\infty}$, $\{\lambda(k)\}_{k=1}^{\infty}$, $\{\mathbf{u}(k)\}_{k=1}^{\infty}$. Such assumption implies that $\liminf_{n \to +\infty} \frac{\nu(i,n)}{n} > 0$, which in turn means that all nodes perform iterations comparably often.

Assumption over $\{\tau_{ij}(k)\}$*:* The simplest assumption is to assume that the delays are bounded (i.e. $\tau_{ij}(k) \in [0, \overline{\tau}]$ for i, j). A more general one is to assume that $\frac{\tau_{ij}(k)}{k} \to 0$ a.s. $\forall i, j$. Alternative assumptions are presented in p.84 of [5].

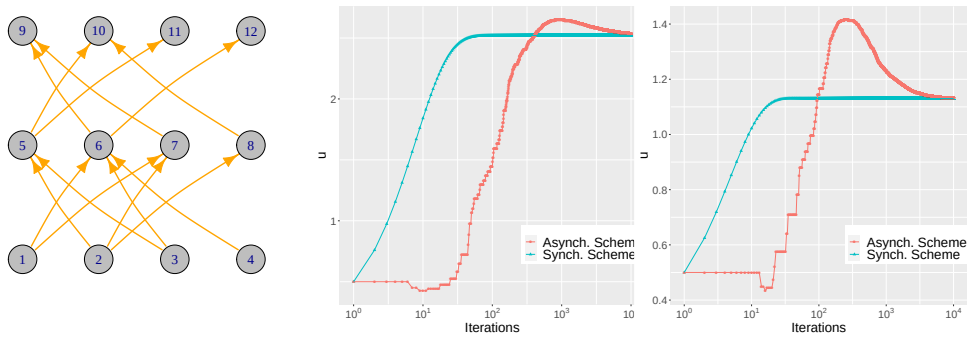
In the following section, we numerically illustrate the convergence of the asynchronous algorithm under the above assumptions.

5 NUMERICAL EVALUATION

In this section, we numerically evaluate the efficiency of the two proposed algorithms. Our goals are to 1) evaluate the convergence speed of the algorithms and, in particular, 2) contrast the synchronous and asynchronous solutions. To that aim, we consider a toy example which is admittedly simple but already serves to illustrate the above points.

We illustrate the application of our two algorithms under the network depicted in Figure 2(a). Motivated by proportional fairness, we let $U_i(x_i) = \alpha_i \log(x_i + 1)$, with $\alpha_i \sim \text{Unif}(0.5, 10)$ for every $i \in \mathcal{I}$. We consider quadratic costs, $C_1(u_i) = \frac{1}{2} u_i^2$, and square-root transfer functions $f_i(x) = \sqrt{x}$, for every $i \in \mathcal{I}$.

Under the asynchronous algorithm, $Y_k \subset \{1, \ldots, I\}$ comprises 4 nodes selected uniformly at random among the 12 nodes. Except for the first 10 iterations, $\delta_{ij}(n)$ is sampled uniformly at random from the set $\{n - 10, \ldots, n\}$ for every $i \neq j$. We also let $\eta = 0.1$, $a(k) = 0.1/(\lfloor k/100 + 1 \rfloor)^{2/3}$,



(a) Network DAG (b) Node 2 (c) Node 11

Fig. 2. Network topology and amount of external resources sent to nodes 2 and 11.

Table 2. Final outputs: amount of external resource sent to each node after 10,000 iterations

Node Id	Initial Value	Final Value	
		Synchronous	Asynchronous
1	0.1434797	3.763868	3.792356
2	0.3456672	2.521142	2.516035
3	0.9293197	2.107925	2.097251
4	0.7872189	2.717501	2.723434
5	0.6156241	2.738282	2.768356
6	0.9371252	1.602584	1.611982
7	0.8965552	2.383430	2.393628
8	0.4284343	1.358760	1.329985
9	0.3854771	0.9696633	0.9795704
10	0.3968619	1.5792735	1.5888138
11	0.5107909	1.1303434	1.1327786
12	0.7000209	1.1976334	1.2010990

$b(k) = 0.1/(\lfloor k/100+1 \rfloor)$ and $c(k) = 0.1/(\lfloor k/100+1 \rfloor \log(\lfloor k/100+1 \rfloor)+1)$. To simplify presentation, we assume that the clocks of all node are in sync, i.e., $\nu(i,k) = \nu(j,k)$. Table 2 indicates that both algorithms converge to the same solution, whereas Figure 2 suggests that convergence occurs at different speeds. The synchronous algorithm converges in 100 iterations, whereas the asynchronous algorithm does so after 2,000 iterations. The outputs of the two algorithms are nearly the same after 10,000 iterations.

Note that although in this simple example the convergence of the synchronous algorithm occurs first, the computational cost per iteration of the synchronous algorithm is at least twice of its asynchronous counterpart. Indeed, less than half of the nodes are performing computation at each iteration of the asynchronous algorithm. In addition, as pointed in Section 4 the asynchronous algorithm is particularly suited for scenarios in which certain nodes may fail or linger inactive for a while. In those cases, the asynchronous algorithm will still allow other nodes to continuously evolve, whereas the synchronous algorithm convergence is utterly bounded by the speed of the slowest node.

6 CONCLUSION

Inspired by backpropagation for neural network training, we have designed and analyzed a backpropagation approach for resource allocation under the NUM framework. In the proposed synchronous algorithm, gradients are computed at leaves and backpropagated upstream. Then, we have extended the idea to an asynchronous setting, also establishing convergence properties. The two algorithms are distributed and suitable for continued operation. In this short paper, we focused on a DAG topology, but the asynchronous algorithm naturally extends to general topologies. In particular, as long as (2) is a contraction map, the asynchronous algorithm will converge to local optima of (3).

Our work is a first step in the use of backpropagation for resource allocation purposes, and opens up a number of different avenues for future research. Among those, we envision a formal analysis of the convergence rate of the algorithms, including conditions under which the asynchronous algorithm is faster than its synchronous counterpart, e.g., when accounting for node failures and repair times. We also envision extensions to account for Byzantine nodes that do not follow the proposed algorithms. Then, a key challenge consists in determining to what extent an adapted version of the algorithms can remain robust against such misbehaving nodes.

REFERENCES

[1] D. Acemoglu, A. Ozdaglar, and A. Tahbaz-Salehi. Networks, shocks, and systemic risk. Technical report, National Bureau of Economic Research, 2015.
[2] A. S. Bedi, A. Koppel, and K. Rajawat. Asynchronous decentralized stochastic optimization in heterogeneous networks. *arXiv preprint arXiv:1707.05816*, 2017.
[3] A. S. Bedi, A. Koppel, and K. Rajawat. Asynchronous saddle point method: Interference management through pricing. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 3229–3235. IEEE, 2018.
[4] D. P. Bertsekas. *Network optimization: continuous and discrete models*. Athena Scientific Belmont, 1998.
[5] V. S. Borkar. *Stochastic approximation: a dynamical systems viewpoint*, volume 48. Springer, 2009.
[6] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
[7] H. Kushner and G. G. Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.
[8] C. Li, X. Yu, T. Huang, and X. He. Distributed optimal consensus over resource allocation network and its application to dynamical economic dispatch. *IEEE transactions on neural networks and learning systems*, 29(6):2407–2418, 2017.
[9] M. J. Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.
[10] A. Ozdaglar and R. Srikant. Incentives and pricing in communication networks. *Algorithmic Game Theory*, 647:571–591, 2007.
[11] P. Parag, S. Sah, S. Shakkottai, and J.-F. Chamberland. Value-aware resource allocation for service guarantees in networks. *IEEE Journal on Selected Areas in Communications*, 29(5):960–968, 2011.
[12] S. Pu, W. Shi, J. Xu, and A. Nedic. Push-pull gradient methods for distributed optimization in networks. *IEEE Transactions on Automatic Control*, 2020.
[13] S. M. Shah and V. S. Borkar. Distributed stochastic approximation with local projections. *SIAM Journal on Optimization*, 28(4):3375–3401, 2018.
[14] T. Shu, M. Liu, Z. Li, and Q. Wu. Interference pair-based distributed spectrum allocation in wireless mesh networks with frequency-agile radios. In *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 233–241. IEEE, 2011.
[15] E. Stai and S. Papavassiliou. User optimal throughput-delay trade-off in multihop networks under num framework. *IEEE Communications Letters*, 18(11):1999–2002, 2014.
[16] T. K. Vu, M. Bennis, M. Debbah, and M. Latva-Aho. Joint path selection and rate allocation framework for 5g self-backhauled mm-wave networks. *IEEE Transactions on Wireless Communications*, 18(4):2431–2445, 2019.
[17] E. Wei, A. Ozdaglar, and A. Jadbabaie. A distributed newton method for network utility maximization—part ii: Convergence. *IEEE Transactions on Automatic Control*, 58(9):2176–2188, 2013.
[18] B. Ying, K. Yuan, and A. H. Sayed. Supervised learning under distributed features. *IEEE Transactions on Signal Processing*, 67(4):977–992, 2018.

[19] J. Zhang, D. Zheng, and M. Chiang. The impact of stochastic noisy feedback on distributed network utility maximization. *IEEE Transactions on Information Theory*, 54(2):645–665, 2008.