Asynchronous Subgradient Push: Fast, Robust, and Scalable Multi-Agent Optimization

Mahmoud S. Assran

Master of Engineering

Department of Electrical and Computer Engineering

McGill University

Montreal, Quebec

April 10, 2018

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Master of Engineering

ACKNOWLEDGEMENTS

I would like to start off by thanking Mike, my advisor and mentor, who introduced me to optimization and machine learning while I was still pursing my undergraduate degree. Your guidance and support has been incontrovertibly invaluable, in more ways than one — not least of all for all your editorial and advisory efforts in all research culminating in this thesis — the list runs the gamut, and I'm eternally grateful that our paths have crossed. The research process is often frustrating, but ultimately greatly satisfying. Thank you to my dear friends Kevin, Stephan, Charlotte, Paul, Reid, Oyku, and Jonathan, for helping me get through those frustrating times. Your friendship has been a tremendously precious support throughout. Most of all, thank you to my family for helping me through it all and for all the love and effort you've invested in me throughout my life; I owe any and all of my success to you.

ABSTRACT

The need to develop distributed optimization methods is rooted in practical applications involving the processing of data that is naturally distributed, private, or simply too large to store on a single machine. In the past decade, a large number of distributed algorithms for solving large-scale convex optimization problems have been proposed and analyzed in the literature, especially from the perspective of multi-agent systems. Although it is fairly well understood which algorithms have the most desirable theoretical properties, many of the theoretical analyses ignore important practical issues such as asynchronism and communication delays. As a result, it is often the case that algorithms with the most desirable theoretical properties (*e.g.*, fastest convergence rates in iterations) do not necessarily have the most desirable properties in practice (*e.g.*, fastest convergence rates in time). Based on this observation, we propose a new distributed optimization algorithm termed Asynchronous Subgradient-Push. Through numerical experiments we demonstrate that Asynchronous Subgradient-Push converges faster than the state-of-the-art multi-agent methods in practice, is more robust to failing/stalling agents, and scales better with the network size. Motivated by the method's superior empirical performance, we develop a convergence theory, and, in particular, show that a subsequence of the iterates at each agent converges to a neighbourhood of the global minimum, where the size of the neighbourhood depends on the degree of asynchrony in the multi-agent network. We also implement the state-of-the-art first-order methods compared in this work using the MPI (Message Passing Interface) standard for message-passing

in clusters, and make them available to the community. In addition, throughout the process of our analysis we develop some peripheral results concerning an asynchronous version of the Push-Sum algorithm for consensus averaging — a building block for many of the state-of-the-art distributed optimization methods proposed in the literature — that are interesting in their own respect. In particular, we show that agents running the Push-Sum consensus-averaging algorithm asynchronously converge to the average of the network geometrically fast (*i.e.*, at a rate of $\mathcal{O}\left(\lambda^k\right)$), where the constant of geometric convergence, λ, depends on the maximum delay and the connectivity of the communication topology, and this convergence holds even in the presence of exogenous perturbations at each agent that seek to derail the consensus process.

ABRÉGÉ

Le besoin de développer des méthodes d'optimisation distribuées est nécessaire dans des applications pratiques impliquant le traitement de données naturellement distribuées, privées, ou simplement trop grandes à stocker sur une seule machine. Au cours de la dernière décennie, un grand nombre d'algorithmes distribués ont été trouveés pour résoudre des problèmes d'optimisation convexe à grande échelle, en particulier pour les systèmes multi-agents. Même si les algorithmes avec des propriétés théoriques les plus souhaitables sont connus, beaucoup d'analyses théoriques ignorent des problèmes pratiques importants, tels que l'asynchronisme et les délais de communication. Par conséquent, il arrive souvent que ces algorithmes ayant des propriétés théoriques souhaitables (par exemple, un taux de convergence rapide en itérations) n'aient pas nécessairement les propriétés les plus souhaitables en pratique (par exemple, un taux de convergence rapide en temps). Sur la base de cette observation, nous proposons un nouvel algorithme d'optimisation distribuée appelé: "Asynchronous Subgradient-Push". Grâce à des expériences numériques, nous démontrons que "Asynchronous Subgradient-Push" converge plus rapidement que les méthodes multi-agents de pointe dans la pratique, est plus robuste aux agents défaillants / bloquants, et évolue mieux avec la taille du réseau. Motivés par la performance empirique supérieure de la méthode, nous développons une théorie de convergence et démontrons en particulier qu'une sous-séquence des itérations de chaque agent converge vers un voisinage du minimum global, où la taille du voisinage dépend du degré d'asynchronie dans le réseau multi-agent. De plus, nous implémentons les

méthodes de premier ordre en utilisant la norme MPI (Message Passing Interface) pour la transmission de messages dans des grappes de calcul les comparons avec notre travail, et les mettons à la disposition de la communauté. De plus, tout au long du processus de notre analyse, nous développons des résultats périphériques concernant une version asynchrone de l'algorithme "Push-Sum" pour le consensus de moyenne — une élément fondamental de plusieurs des méthodes d'optimisation distribuées de pointe proposées dans la littérature — qui sont intéressants dans leur propre égard. En particulier, nous montrons que les agents exécutant l'algorithme de consensus de moyenne de type "push-sum" convergent à un taux géométrique de façon asynchrone (c'est-à-dire à un taux de $\mathcal{O}\left(\lambda^k\right)$) vers la moyenne du réseau. La convergence géométrique dépend du délai maximum et de la connectivité de la topologie de communication, et cette convergence est toujours valable même en présence de perturbations exogènes de chaque agent qui cherchent à faire dérailler le processus consensuel.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Motivation

Distributed optimization has played an ever-increasing role in society due to the emergence of numerous applications encompassing distributed sensing systems [65], the internet of things [81, 41, 60], the smart grid [68, 42], multi-robot systems [46, 62, 12] and large-scale machine learning [72, 73, 74, 20, 1, 64, 86, 9] to name a few. To say that everything is an optimization problem would be a tautology, and the field of distributed optimization has the potential to touch many seemingly disjoint subject areas all under this premise. Optimization is very powerful in that it allows us to obtain excellent and justifiable solutions to problems once they have been modelled, however the modelling process often involves data, and when the data is naturally distributed, private, or simply too large to store on a single machine, we turn to distributed optimization algorithms to obtain solutions to our problems. The emergence of Big Data has also played a large role in the burgeoning of distributed optimization. In the wake of Big Data, solving a "typical" optimization problem has become an increasingly time-consuming and resource hungry undertaking, and, as a result, we may turn to distributed optimization algorithms simply to obtain faster solutions to our problems. More generally, there has always been a need for the solution of very large computational problems, whether data-based or not, and distributed optimization provides an efficient way to solve these problems [5,

72]. While raw computational throughput and storage capacity have increased at exponential rates as predicted by Moore's Law, transistor efficiencies have plateaued, as dictated by Dennard's law, and so it follows that in order to handle the massive computational and storage resources demanded by Big Data at reasonable power costs, we must increasingly rely on distributed optimization [14]. All of these motives have propelled research in the area of distributed optimization, and, as a result, there have been significant advances in the development of distributed methods with theoretical convergence guarantees. In general, the field of distributed optimization looks at how to perform distributed optimization quickly and robustly.

1.2 Distributed Optimization

There have been many approaches taken towards the design and development of distributed optimization algorithms. The seminal reference of Bertsekas and Tsitsiklis [5] presents many of these methods cogently. Some approaches, under the aegis of parallel optimization, focus on parallelization at the task level using colocated processors [5] (*e.g.*, the optimization problem is somehow broken into several discrete subproblems that can be solved concurrently, and each processor works independently on one of these subproblems) — these methods are interesting in their own right, but do not technically belong to the domain of distributed optimization. Distributed optimization is more generally concerned with having multiple agents (processors) work together to solve an optimization problem, with parallelization typically performed at the data or parameter level [5] (*e.g.*, each agent performs computations using a subset of the data, or only updates a subset of the parameters). In distributed computing systems, processors may be far apart; communication

delays may be unpredictable; communication links may be unreliable; the topology may undergo changes during operation due to failures/repairs of communication links and or processors; and each processor may be engaged in its own private activities while at the same time cooperating with other processors in the context of some computational task [5]. High performance computing clusters fit this model of distribution quite nicely [72], especially since node and link failures have become the norm rather than the exception [74, 39, 19].

Loosely speaking, we can categorize much of the work in the distributed optimization literature according to three criteria. The first criterion is the presence or absence of a global control mechanism. At one extreme, the global control mechanism is only used to load a common program to the processors, and each processor is allowed to work on its own thereafter. At the other extreme, the control mechanism is used to instruct each processor on what to do at each step. The former falls under the category of distributed methods, while the latter falls under the category of centralized methods. Centralized methods are non-ideal for several reasons; the master node becomes a bottleneck on the entire optimization process and a central point of failure; it follows that without the recourse of highly optimized tools, such as MapReduce [21], centralized methods can be highly suboptimal when scaling to large processing systems [14]. Actually, even MapReduce [21], a popular distributed computing tool used in machine learning applications, is known to be ill-suited for the iterative computations inherent in the training of large-models and deep-learning in particular [20]. Decentralized methods bypass many of these issues

by providing increased robustness to node failure, and scalability with the network size [74, 76, 77, 66, 75, 72, 65].

The second criterion for classifying work in the distributed optimization literature concerns the general operating principle of the algorithm: synchronous or asynchronous. The distinction here refers to the presence or absence of a common global clock used to synchronize the operation of the different processors. The chief benefit of synchronous operation is that the behaviour of the processors is much easier to control, and algorithm design is considerably simplified. On the other hand, synchronous operation introduces undesirable overhead that can greatly degrade the algorithm's efficiency in practice [66, 5, 20, 44, 2, 11, 32, 40, 3, 82, 48, 47].

The third criterion is the processor interconnection, the mechanism by which processors exchange information. In a shared memory architecture, such as that utilized by the *Hogwild!* algorithm [69], processors communicate by writing variables to a shared memory, however this architecture necessitates the need for access control to make multiple-access of the same memory block safe. In a message-passing architecture, each processor has its own local memory and processors communicate through an interconnection network consisting of direct communication links joining certain pairs of processors. Even though it would be best if all processors were directly connected to each other, this is often not feasible. Either there is an excessive number of links, which leads to increased cost, or processors communicate through a bus which leads to excessive communication delays due to bus contention.

This thesis is primarily concerned with distributed message-passing systems of both the synchronous and asynchronous variety. Message-passing systems encompass

anything from processors on a single machine communicating over a bus, to servers in a high-performance computing cluster communicating over an InfiniBand network, to sensor-nodes scattered around a city communicating over an ad-hoc network.

1.3 Problem Formulation

In this section we describe the standard formulation of the unconstrained optimization problems arising in the context of *message-passing distributed* optimization algorithms; these algorithms are sometimes referred to as multi-agent methods in the literature.

We are interested in distributed algorithms to solve the optimization problem

$$\underset{x\in\mathbb{R}^d}{\text{minimize}} \quad F(x) \coloneqq \sum_{i=1}^{n} f_i(x) \tag{1.1}$$

where $x \in \mathbb{R}^d$ is the optimization variable, and the functions f_i are strongly-convex with Lipschitz-continuous gradients. We consider multi-agent frameworks consisting of n agents connected together over a communication network. Information about each function f_i (such as a black-box oracle to obtain gradients of f_i) is only available at agent v_i, and the agents must cooperate by communicating over a network in order to find a minimizer. Multi-agent methods are iterative, and each agent maintains a local copy of the decision variable; the ultimate goal is for all agents to agree (*i.e.*, achieve a consensus) on a minimizer $x^\star$ of (1.1). Hence, message-passing distributed optimization algorithms often go by the moniker of *consensus-based* distributed optimization methods.

This problem formulation is quite general and arises in many applications such as robust statistical inference [65], formation control [62], non-autonomous power

control [68], distributed message routing [60], spectrum access coordination [87], and empirical loss minimization in machine learning [72]. For example, each agent can use a subset of the data to construct its own local loss function, and then work together to agree on the model parameters that minimize the sum of their local loss functions.

1.4 Related Work

Consensus-Based Multi-Agent Optimization

One of the earliest related works on multi-agent optimization is that of Nedic and Ozdaglar [52], which analyzes distributed subgradient algorithms using a doubly-stochastic consensus scheme. At each local iteration, each agent performs a local, greedy, optimization step using its local objective function to update its estimate of the optimal model parameters, and then performs a consensus step by gossiping with its neighbours in the communication topology, and, in particular, takes a convex combination of its parameter estimate with those of its neighbours. This method is typically analyzed in matrix-form by stacking all of the agents' parameter vectors into a single parameter-matrix; the special convex combination of the agents' estimates — the consensus procedure — can then be represented by multiplying the parameter-matrix by a so-called *consensus-matrix*, which conforms to the graph structure of the communication topology, and has stochastic rows and columns (*i.e.*, all entries are nonnegative, and the rows and columns sum to 1). Consensus-matrices with stochastic rows and columns are referred to as doubly stochastic. Similar works such as [56, 54] use a similar approach in mixing a local subgradient iteration with a doubly-stochastic averaging step.

Consensus Averaging in Multi-Agent Optimization

In fact, most multi-agent optimization methods build on distributed averaging algorithms [72]. However, it turns out that doubly stochastic protocols are undesirable for many reasons, especially in peer-to-peer networks, which tend to lack a highly organized structure. This lack of organization as well as the fact that node and link failures have become the norm rather than the exception [74, 39, 19] present new types of constraints on the consensus algorithms being used, thereby eliminating the applicability of many doubly-stochastic protocols, which in general tend to have more strict network constraints [74, 39, 75, 58, 88, 57]. To eliminate the need for doubly stochastic averaging, the Push-Sum approach for consensus averaging, which only requires singly-stochastic consensus-matrices, was introduced in [39], where it was also analyzed in the case of fully connected communication graphs. The analysis was extended in [4] for general connected graphs. Further work has provided convergence guarantees in the face of the other practical issues, such as communication delays and dropped messages [16, 30, 15, 29]. In general, Push-Sum is attractive for implementations because it can easily handle directed communication topologies, and thus avoids incidents of deadlock that may occur in practice when using undirected communication topologies [74].

Multi-Agent Optimization with Singly-Stochastic Consensus Matrices

Rabbat and Tsianos proposed and analyzed the first multi-agent optimization algorithm using Push-Sum for distributed averaging [75] — they studied a variant of the Distributed Dual Averaging algorithm [25]. Nedic and Olshevsky continued this line of work by proposing and analyzing the Subgradient-Push method [57], a

distributed (sub)gradient algorithm that uses the Push-Sum protocol in place of the original doubly-stochastic consensus procedure. Yin and Zeng, and Khan and Xi simultaneously came out with the DEXTRA and Extra-Push algorithms for multi-agent optimization using the Push-Sum protocol; their algorithms are able to achieve geometric convergence rates over directed graphs [85, 84, 89, 88]. Nedic, Olshevsky, and Shi built upon this work by proposing the Push-DIGing algorithm for multi-agent optimization using the Push-Sum protocol, which is able to achieve a geometric convergence rate over directed and time-varying communication graphs [49]. The Push-DIGing and DEXTRA/Extra-Push algorithms are considered to be the state-of-the-art, and the Subgradient-Push algorithm, a multi-agent analog of classical gradient descent, is considered a baseline method. It should be noted that all of these algorithms are *synchronous* in nature.

Asynchronous Multi-Agent Optimization

Most recently, in the past year or so, there have been several asynchronous multi-agent optimization algorithms proposed in the literature, such as [83], which requires doubly-stochastic consensus over undirected graphs; [26], which requires push-pull based consensus over undirected graphs; and [50], which assumes a model of asynchrony in which agents become activated, or "wake-up," according to a Poisson point process and "finish updates before another agent becomes activated."

The seminal work on asynchronous multi-agent optimization algorithms of Tsitsiklis et al. [78] considers the case where each agent holds one component (or block) of the optimization variable, and can locally evaluate the gradient of the global objective with respect to its component. Convergence is proved for a distributed gradient

algorithm in that setting, but that setting is also inherently different from the proposed problem formulation where each agent does not necessarily have access to the global objective. The work of Li and Basar [43] studies distributed asynchronous algorithms and proves convergence and asymptotic agreement in a stochastic setting, but assumes a similar computation model to that of Tsitsiklis et al. [78] in which each agent updates a portion of the parameter vector using an operator which produces contractions with respect to the global objective.

As was already mentioned, there exist non-consensus-based parallel asynchronous optimization algorithms, such as [69, 63, 33], which are implemented on either a shared-memory multiprocessor system, or utilize some sort of parameter server. These methods are actually quite successful at performing certain tasks in practice, however, they represent a fundamentally different computing architecture from the multi-agent setting considered in this work.

In general, many of the asynchronous multi-agent optimization algorithms proposed in the literature make restrictive assumptions regarding the nature of the agent updates (*e.g.*, sparse Poisson point process [50], randomized single activation [8, 24], randomized multi-activation [38, 27, 53, 37, 22, 79, 36, 7, 90]). To the best of our knowledge, there are no asynchronous multi-agent optimization algorithms that utilize singly-stochastic consensus and don't make these types of assumptions.

Theory-Practice Knowledge Gap

It is interesting to note that, in the past decade, a large number of distributed algorithms for solving large-scale convex optimization problems have been proposed and analyzed in the literature, especially from the perspective of multi-agent systems.

Although it is fairly well understood which algorithms have the most desirable theoretical properties, there has been relatively little work investigating and evaluating practical implementations of these algorithms and there is a non-trivial gap between theory and practice [34, 23, 76, 72, 74, 73]. For example, many of the theoretical analyses ignore important practical issues such as asynchronism and communication delays. As a result, it is often the case that algorithms with the most desirable theoretical properties (*e.g.*, fastest convergence rates in iterations) do not necessarily have the most desirable properties in practice (*e.g.*, fastest convergence rates in time).

1.5 Contributions

In this work we extend the Subgradient-Push optimization algorithm to asynchronous operation and term the extended algorithm Asynchronous Subgradient-Push. Through numerical experiments we demonstrate that Asynchronous Subgradient-Push converges faster than the state-of-the-art multi-agent methods in practice, is more robust to failing/stalling nodes, and scales better with the network size [2]. Motivated by the superior empirical results of the proposed algorithm, we proceed to develop a convergence theory: when the local objective functions are strongly convex with Lipschitz-continuous gradients, we show that a subsequence of the iterates at each agent converges to a neighbourhood of the global minimum, where the size of the neighbourhood depends on the maximum delay, the modulus of strong-convexity, and the Lipschitz constant. In addition, throughout the process of our analysis we develop some peripheral results concerning an asynchronous version of the Push-Sum Protocol used for consensus averaging that are interesting in their own respect.

In particular, we show that agents running the Push-Sum Protocol asynchronously converge to the average of the network — even in the presence of exogenous perturbations at each agent that seek to derail the consensus process — geometrically fast, where the constant of geometric convergence depends on the consensus-matrices' degree of ergodicity (cf. Hajnal and Bartlett [31]), and takes into account the degree of asynchrony in the multi-agent network. We also implement the state-of-the-art first-order methods compared in this work using the Message Passing Interface (MPI) standard for message-passing in clusters [18, 17, 28], and make them available to the community.[1]

1.6 Proposed Algorithm & Main Results

In this work we are particularly interested in *asynchronous* multi-agent methods, by which we mean that agents do not necessarily perform updates at the same times or at the same rate, and that messages between agents may be subject to delays. Agents do not wait for each other to complete computations, nor do they wait for messages to be received before moving on to the next step in the algorithm; this asynchronous nature makes it possible for agents to perform a drastically different number of gradient steps over any time interval.

Practical asynchronous implementations of multi-agent communication using the Message Passing Interface (MPI) [28], or other message passing standards, often have the notion of a *send-buffer* and a *receive-buffer*. A send-buffer is a data structure containing the messages sent by an agent, but not yet physically transmitted by the

[1] Open Source: https://github.com/MidoAssran/maopy

Algorithm 1 Asynchronous Gradient Push (Pseudocode) for agent v_i

Initialize $x \leftarrow$ some vector in $\mathbb{R}^n$ {Push-sum numerator}
Initialize $y \leftarrow 1$ {Push-sum weight}
Initialize $\alpha \leftarrow$ some scalar value in $\mathbb{R}_{++}$ {Step-size}
$N^{\text{out}} \leftarrow$ number of out-neighbours
repeat
  begin: Local Computation
  $z \leftarrow x/y$ {Debias consensus estimate of minimizer}
  $x \leftarrow x - \alpha \nabla f_i(z)$ {Greedy local minimization of $f_i(\cdot)$}
  Update step-size α
  end
  begin: Asynchronous Gossip
  Copy message $m = (x/N^{\text{out}}, y/N^{\text{out}})$ to local send-buffer {Send messages to neighbours}
  $x, y \leftarrow x, y +$ sum of all messages in local receive-buffer {Process received messages}
  end
until *termination.*

underlying communication system. A receive-buffer is a data structure containing the messages received by an agent, but not yet processed. Using this notion of send- and receive-buffers, the pseudocode running on each individual agent in the network is provided in Algorithm 1 and entails a basic two-step procedure consisting of **Local Computation** followed by **Asynchronous Gossip**. During the **Local Computation** phase, agents update their estimate of the minimizer by performing a local (sub)gradient-descent step. During the **Asynchronous Gossip** phase, agents copy all outgoing messages into their local send-buffer and subsequently process (sum) all messaged received (buffered) in their local receive-buffer while the agent was busy performing the preceding **Local Computation**. The underlying communication system begins transmitting the messages in the send-buffer once they are copied there; thereby freeing the agent to proceed to the next step of the algorithm without waiting for the messages to reach their destination.

Some brief notation that we use to state our main result: for an agent running the code in Algorithm 1, we denote the value of the variable z held locally by agent v_i at time $t[k]$ by $z_i[k]$, and we generalize this notation to other variables as well. We also represent the communication-topology as a directed graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes (agents), and $\mathcal{E}$ is the set of edges (directed communication links). Let $\overline{\tau}^{\text{proc}}$ ($\in \mathbb{Z}_{++}$, positive, non-zero, integers), denote a measure of the asynchrony in the multi-agent network; we formalize $\overline{\tau}^{\text{proc}}$ in our analysis. For ease of exposition, we assume that the communication-graph is static and strongly-connected. The strongly-connected property of the directed graph is necessary to ensure that all agents are capable of influencing each other's values, and in the discussion section we describe how one can extend our analysis to account for time-varying directed communication-topologies.

1.6.1 Constant Step-Size

If the agents run Algorithm 1 with a constant step-size, α, which satisfies an appropriate upper-bound, and the local objective functions are strongly-convex with Lipschitz-continuous gradients, then for all agents, v_i, it holds that

$$\liminf_{k\to\infty} \|z_i[k] - x^\star\| < \frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)} + \mathcal{O}\left(\frac{\alpha}{1-q}\right),$$

where $x^\star$ is a globally optimal solution to problem (1.1), q depends on consensus-matrices' degree of ergodicity taking into account the delays, and L and m are the Lipschitz constant and modulus of strong-convexity, respectively.

The $\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)}$ term is due to the asynchrony in the multi-agent system. If $\overline{\tau}^{\text{proc}}$ decays to 1, that is, if the algorithm operates *semi-synchronously*

(agents wait for each other to complete updates, but don't wait for messages to be sent/received), then this term completely disappears. The $\mathcal{O}\left(\alpha/(1-q)\right)$ term is due to the consensus-disagreement between agents. If the agents have little influence on each other (large $1/(1-q)$), or if the agents use a large step-size, meaning it's easier for them to move away from consensus in their local optimization steps, then the size of the neighbourhood to which they are guaranteed to converge also increases.

1.6.2 Diminishing Step-Size

If the agents run Algorithm 1 with a diminishing step-size and the local objective functions are strongly-convex with Lipschitz-continuous gradients, then for all agents, v_i, it holds that

$$\liminf_{k\to\infty} \|z_i[k] - x^\star\| < \frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)},$$

where, once again, $x^\star$ is a globally optimal solution to problem (1.1), and L and m are the Lipschitz constant and modulus of strong-convexity, respectively. If $\overline{\tau}^{\text{proc}}$ decays to 1, that is, if the algorithm operates *semi-synchronously* (agents wait for each other to complete updates, but don't wait for messages to be sent/received), then the parameter estimates, z, at each agent converge to the same globally optimal solution. In order to implement such a diminishing step-size, the agents need not know the exact times at which other agents perform updates, instead they can use the technique suggested in [74] to update their step-size using actual wall-clock time. For example, each agent can maintain a counter, k, that increments by 1 every 100ms or so. This technique works in practice and has been validated through numerous experimental results [74, 72].

1.7 Thesis Overview

In Chapter 2, a detailed background regarding some of the terminology and machinery commonly used in the convex analysis and optimization literature is provided and accompanied by a formal description of the Push-Sum consensus-averaging algorithm and the Subgradient-Push, Push-DIGing, and Extra-Push optimization algorithms. In Chapter 3 we describe the System Model constructed for the analysis of asynchronous algorithms. In Chapter 4 we develop an asynchronous formulation of the Push-Sum protocol and prove a geometric convergence rate in the asynchronous setting, even in the presence of exogenous perturbations that seek to derail the consensus process at each agent. In Chapter 5 we develop an asynchronous formulation of the Subgradient-Push algorithm (termed Asynchronous Subgradient Push), provide numerical experiments showcasing its superior performance relative to the state-of-the-art methods in the literature, and, using the results from Chapter 4, prove convergence to a neighbourhood of the global minimum, where the size of the neighbourhood depends on the degree of asynchrony in the multi-agent network. In Chapter 6 we summarize our findings and describe possible extensions to future work.

CHAPTER 2
Background

In this chapter we describe in a bit more detail some of the related work on distributed optimization methods in the literature and introduce the Push-Sum protocol for consensus-averaging formally. We also introduction to some of the terminology commonly used in the convex analysis and distributed optimization literature.

Notation. Let $\mathbf{I}_n$ denote the $n \times n$ identity matrix, and $\mathbf{1}_n$ represent an n-dimensional column vector with each entry equal to 1. A generic local variable z held by agent v_i in the network at time-index k is denoted by the vector $z_i[k]$, where $z_i[k] \in \mathbb{R}^d$. Let $\mathbf{z}[k] \in \mathbb{R}^{n \times d}$ be a matrix which stores the copy of variable z for the entire n-agent network at time-index k, where $(z_i[k])^T$ is the i^{th} row of $\mathbf{z}[k]$. In general, all matrices are represented with boldfaced symbols, and all vectors with regular math font, with the exception of $\mathbf{1}_n$ which is a vector.

2.1 Convex Optimization & Graph Theory

Convexity. A scalar-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y),$$

for some $\theta \in [0, 1]$ and $x, y \in \mathbb{R}^d$. The principal benefit of convex optimization (versus non-convex optimization) is that local optimization and global optimization become one task. That is, any local minimum found is also guaranteed to be a global minimum. Non-convex optimization on the other hand makes no such guarantees,

and as such, non-convexity is generally believed to increase the difficulty of the problem at hand [10].

Gradient and Subgradient. The gradient of a smooth convex function $f : \mathbb{R}^d \to \mathbb{R}$ at $x \in \mathbb{R}^d$ is a unique vector $g \in \mathbb{R}^d$ which satisfies the (sub)gradient inequality

$$g^T(y - x) \leq f(y) - f(x),$$

for all $y \in \mathbb{R}^d$. The entries of the gradient g correspond to the partial derivatives of f at x, and so g is often denoted by $\nabla f(x)$. If f is non-smooth, *i.e.*, there exist several vectors g that satisfy the (sub)gradient inequality (note, this is not a formal definition of smoothness), then the vectors g are called the subgradients of f at x, and the set of all subgradients g is called the subdifferential of f at x, denoted by $\partial f(x)$.

First Order Optimality. If $x^\star$ is a minimizer of a smooth scalar-valued convex function $f : \mathbb{R}^d \to \mathbb{R}$, then the gradient of f at $x^\star$ is the zero-vector $\mathbf{0} \in \mathbb{R}^d$. Alternatively, if f is non-smooth, then the zero-vector $\mathbf{0}$ is in the subdifferential of f at $x^\star$, $\partial f(x^\star)$.

Strong Convexity. A scalar-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is m strongly-convex, with modulus of strong convexity m ($\in \mathbb{R}_{++}$, positive, non-negative, reals), if and only if the function $x \mapsto f(x) - \frac{m}{2} \|x\|^2$ is convex. Strong convexity gives a lower bound on the growth of a function [10], thereby allowing us to reliably use the closeness of the gradient $\nabla f(x)$ to $\mathbf{0}$ as a measure of the closeness of x to the minimizer (if the minimizer exists). Given that a whole class of iterative optimization

algorithms, under the aegis of *first-order methods*, rely on the gradient to determine the minimizer of a function, it comes as no surprise that strong-convexity is a relatively standard assumption in the literature.

Lipschitz Smoothness. A scalar-valued convex function $f : \mathbb{R}^d \to \mathbb{R}$ is M Lipschitz-smooth if its gradients are M Lipschitz-continuous; *i.e.*, there exists a constant $M \in \mathbb{R}_{++}$ such that

$$\|\nabla f(x) - \nabla f(y)\| \leq M \|x - y\|,$$

for all $x, y \in \mathbb{R}^d$. Lipschitz-smoothness ensures that the gradients of a function cannot arbitrarily explode in any direction, a standard assumption in the literature.

For example, the prototypical quadratic

$$f : x \in \mathbb{R}^d \mapsto x^T A x + b^T x + c,$$

for $c \in \mathbb{R}$, $b \in \mathbb{R}^d$, and symmetric positive definite $A \in \mathbb{S}^{d \times d}_{++}$, is strongly convex and Lipschitz-smooth. Another example of a strongly-convex and Lipschitz-smooth function is the regularized cross-entropy loss function

$$f : x \in \mathbb{R}^d \mapsto \sum_{j=1}^{n} ln\left(1 + e^{-t_j w_j^T x}\right) + \frac{\lambda}{2} \|x\|^2,$$

for $w_j \in \mathbb{R}^d$, $t_j \in \{-1, +1\}$, and $n, \lambda > 0$. This function arises quite often in machine learning classification applications — *e.g.*, when performing maximum-likelihood with a logistic-regression likelihood model.

More generally, some examples of functions that are strongly-convex and Lipschitz-smooth are the family of functions of the form

$$f : x \in \mathbb{R}^d \mapsto g(x) + \frac{\lambda}{2} \|x\|^2$$

where $g(x) : \mathbb{R}^d \to \mathbb{R}$ is a (potentially non-convex) function that belongs to the set of twice differentiable functions with bounded Hessian (absolute row sum is bounded), and $\lambda > 0$ is a sufficiently large scalar regularizer (λ greater than the maximum absolute row sum of the Hessian of g at all points x).

Directed Strongly Connected Graphs. A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ consists of a set of vertices, $\mathcal{V}$, and a set of edges, $\mathcal{E}$. Each edge in the edge-set joins a pair of vertices together. The graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is termed directed if there is a direction associated with the edges in the edge-set, and is termed undirected otherwise. A directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is strongly connected if every vertex is reachable from every other vertex by traveling along a directed path adhering to the direction of the edges in the edge-set. If every vertex in the directed graph is reachable from every other vertex by traveling along an *undirected* path, not necessarily adhering to the direction of the edges in the edge-set, then the directed graph is simply referred to as *connected*. A strongly-connected component of a directed graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, is a subgraph that is itself strongly connected, and has the property that no additional edges from $\mathcal{G}(\mathcal{V}, \mathcal{E})$ can be added to the subgraph without breaking its (the subgraph's) strong connectivity.

Graph Conformance. The adjacency matrix of a directed graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, is a $|\mathcal{V}| \times |\mathcal{V}|$ matrix, $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$, that contains a 1 in its $(i, j)^{th}$ entry if there is an edge from vertex i to vertex j, and contains a 0 in that entry otherwise. Note that

the adjacency matrix of a directed graph is *not* necessarily symmetric. An arbitrary matrix $\boldsymbol{P}$ is said to be graph conformant with respect to the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ if $\boldsymbol{P}$ has the same zero/non-zero structure as the adjacency matrix of graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$.

Stochastic, Indecomposable, Aperiodic. A square matrix $\boldsymbol{P}$ is said to be (column) stochastic if all its entries are non-negative, and all the columns sum to 1 (*i.e.*, $\mathbf{1}^T \boldsymbol{P} = \mathbf{1}^T$). A square matrix $\boldsymbol{P}$ is said to be (column) stochastic, indecomposable, and aperiodic (SIA) if it is (column) stochastic, and if

$$\lim_{n \to \infty} \boldsymbol{P}^n = \boldsymbol{Q}$$

exists and all the columns of $\boldsymbol{Q}$ are the same. Furthermore, if the square matrix $\boldsymbol{P}$ is graph conformant with respect to a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, then the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is connected and has only one strongly-connected component. If the square matrix $\boldsymbol{P} \in \mathbb{R}^{v \times v}$ is doubly stochastic, then all of its entries are non-negative, and all the columns *and rows* sum to 1. If $\boldsymbol{P}$ is (doubly) stochastic, indecomposable, and aperiodic, then

$$\lim_{n \to \infty} \boldsymbol{P}^n = \frac{1}{v} \mathbf{1}\mathbf{1}^T.$$

Linear (Geometric) Convergence. A sequence $\{x[k]\}$ is said to converge Q-linearly to a limit point $x^\star$ if

$$\frac{\|x[k+1] - x^\star\|}{\|x[k] - x^\star\|} = \lambda,$$

for all k sufficiently large and some non-negative constant $\lambda \in (0, 1)$. The prefix "Q" stands for "quotient" since this type of convergence is defined in terms of a quotient

of successive error terms [61]. For instance, the sequence $1 + (0.2)^k$ converges Q-linearly to 1 with rate $\lambda = 0.2$. A sequence $\{x[k]\}$ is said to converge R-linearly (geometrically) to a limit point $x^\star$ if there is a sequence of nonnegative scalars $\{v_k\}$ such that

$$\|x[k] - x^\star\| \leq v_k,$$

for $k \in \mathbb{N}$, and $\{v_k\}$ converges Q-linearly to zero. The prefix "R" stands for "root" and characterizes the overall rate of decrease of the error, rather than the decrease between successive steps, and hence represents a slightly weaker form of convergence than Q-linear convergence. For instance, the sequence

$$x[k] = \begin{cases} 1 + (0.2)^k, & k \text{ even}, \\ 1, & k \text{ odd}, \end{cases}$$

is dominated by the sequence $1 + (0.2)^k$ and thus converges R-linearly to 1. Notice that the error $\|x[k] - 1\|$ does not decrease at each step, hence this sequence does not converge Q-linearly.

Sublinear Convergence. A sequence $\{x[k]\}$ is said to converge sublinearly (or sub-geometrically) to a limit point $x^\star$ if

$$\frac{\|x[k+1] - x^\star\|}{\|x[k] - x^\star\|} = \lambda[k],$$

for $k \in \mathbb{N}$, where $\lambda[k] \in (0, 1)$ and $\lambda[k] \uparrow 1$ as $k \to \infty$.

2.2 Synchronous Push Sum Averaging

In this section we describe the Push-Sum algorithm used for consensus averaging in multi-agent networks, a fundamental building block of the first-order state-of-the-art methods compared in this work. Each agent v_i in the n agent network holds a local variable $z_i[0] \in \mathbb{R}^d$. The goal of distributed gossip averaging is to have all the agents in the network agree (achieve consensus) on the average of their initial values by gossiping with their neighbours (*i.e.*, $\boldsymbol{z}[k] \to \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top \boldsymbol{z}[0]$). Converging to the true average of the network through gossip is an iterative process; a typical consensus averaging protocol solves the problem by performing the iterations $\boldsymbol{z}[k+1] = \boldsymbol{P}_0\boldsymbol{z}[k]$ starting at $\boldsymbol{z}[0]$, where $\boldsymbol{P}_0$ is a doubly stochastic SIA matrix. However, as was already mentioned, doubly stochastic protocols are undesirable for many reasons, especially in peer-to-peer networks, which tend to lack a highly organized structure. The Push-Sum protocol achieves consensus averaging using only column stochastic matrices, and is directly used by many multi-agent optimization algorithms. At all time-indices k, each agent v_i locally maintains the variables $w_i[k], z_i[k] \in \mathbb{R}^d$, and $y_i[k] \in \mathbb{R}$. A small subtlety to note is that since each $y_i[k]$ is a scalar, the state of the variable y in the entire network is represented by a vector, $y[k] \in \mathbb{R}^n$, as opposed to a matrix, where $y_i[k]$ is the i^{th} entry of $y[k]$. In order to describe the algorithm from a global perspective, we use the matrix-based formulation of the Push-Sum Averaging protocol provided in Algorithm 2, where $\text{diag}(y)$ is a diagonal matrix with the elements of the vector y on the diagonals. The initializations are $w_i^{(0)}[0] \in \mathbb{R}^d$, and $y_i^{(0)}[0] = 1$. Since the mixing matrices are only column-stochastic, sending agents control their own respective columns of the mixing matrices independently, without

Algorithm 2 Delay Free Synchronous Push-Sum Averaging (cf. [39])

for $k = 0, 1, 2, \ldots$ **to** termination **do**

$$\boldsymbol{w}[k+1] = \boldsymbol{P}_0 \boldsymbol{w}[k] \tag{2.1}$$
$$y[k+1] = \boldsymbol{P}_0 y[k] \tag{2.2}$$
$$\boldsymbol{z}[k+1] = (\mathrm{diag}(y[k+1]))^{-1} \boldsymbol{w}[k+1] \tag{2.3}$$

coordination with the other agents in the network. At each time-index k, each agent v_j gossips (or *pushes*) its scaled push-sum numerator, $[\boldsymbol{P}_0]_{ij}\, w_j[k]$, and its scaled push-sum weight, $[\boldsymbol{P}_0]_{ij}\, y_j[k]$, to its neighbouring agents $\{v_i \in N_j^{\text{out}}\}$. After pushing values to peers, agents subsequently perform a local update by *summing* the push-sum messages that they have received. Any bias built up in the push-sum numerator, $w_j[k]$, is also built up in the scalar push-sum weight, $y_j[k]$, and so a division of the push-sum numerator by the scalar yields the unbiased consensus estimate of the network average. After several such iterations, the consensus estimates z_i[k] at each agent converge to the true average of the network (asymptotically). The column stochastic mixing matrices are typically given by

$$[\boldsymbol{P}_0]_{ij} \coloneqq \begin{cases} \frac{1}{N_j^{\text{out}}}, & (i,j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

That is, each agent gossips with all of its out-neighbours at each iteration with equal weight. In order to do so, the agents must know their number of out-neighbours. Define the "type" of a matrix to be its zero/non-zero structure. Observe that, by definition, the matrix $\boldsymbol{P}_0$ is of the same type as the adjacency matrix of the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, and so the consensus matrix $\boldsymbol{P}_0$ is said to *conform* to the graph structure

Algorithm 3 Synchronous Subgradient-Push (cf. [57])

for $k = 0, 1, 2, \ldots$ **to** termination **do**

$$\boldsymbol{w}[k+1] = \boldsymbol{P}_0 \left(\boldsymbol{w}[k] - \alpha[k]\nabla \boldsymbol{F}[k]\right) \tag{2.5}$$
$$y[k+1] = \boldsymbol{P}_0 y[k] \tag{2.6}$$
$$\boldsymbol{z}[k+1] = \operatorname{diag}(y[k+1])^{-1}\boldsymbol{w}[k+1] \tag{2.7}$$

$\mathcal{G}(\mathcal{V}, \mathcal{E})$. Each agent running the synchronous Push-Sum protocol converges to the network-wide average R-linearly, where the constant of geometric convergence is given by the second-largest eigenvalue of the consensus-matrix $\boldsymbol{P}_0$. Namely, for all $i = 1, 2, \ldots, n$,

$$\left\| z_i[k] - \frac{1}{n}\mathbf{1}_n^T \boldsymbol{w}[0] \right\|_1 \leq C\lambda^k,$$

where $\lambda \in [0, 1)$ is the second-largest eigenvalue of the consensus-matrix $\boldsymbol{P}_0$.

2.3 Synchronous Subgradient Push

The synchronous Subgradient-Push optimization algorithm is used as a baseline and corresponds to a *push-based* analogue of the classical gradient descent algorithm (cf. [61] for a gradient descent reference). At all time-indices k, each agent v_i locally maintains the variables $w_i[k], z_i[k] \in \mathbb{R}^d$, and $y_i[k] \in \mathbb{R}$. In order to describe the algorithm from a global perspective, we use the matrix-based formulation of the synchronous Subgradient-Push method provided in Algorithm 3. The consensus-matrices $\boldsymbol{P}_0$ are as defined in (2.4). The matrix-valued function $\nabla \boldsymbol{F}[k]$ is the Jacobian of the global objective $F(\cdot)$ at $\boldsymbol{z}[k]$. In the non-differentiable but convex objective setting, the rows of $\nabla \boldsymbol{F}[k]$ are members of the subgradient set of the global objective $F(\cdot)$ evaluated at $\boldsymbol{z}[k]$. The initializations are $\boldsymbol{z}[0] = \boldsymbol{w}[0] \in \mathbb{R}^d$, $y[0] = \mathbf{1}_n$, and $\alpha[\cdot]$ is

just a positive scalar step-size. For the agents to achieve consensus and converge to a global minimum, it is recommended that the step-size satisfy the Robbins-Monro conditions ($\sum_{k=0}^{\infty} \alpha[k] = \infty$, $\sum_{k=0}^{\infty} (\alpha[k])^2 < \infty$). The push-sum modification simply interleaves a push-sum averaging step with a gradient-descent step. The push-sum averaging step is used to steer the nodes' estimates of the optimal parameter setting towards each other (consensus), and the gradient-descent step is used to steer the nodes' arguments towards the minimizer (greedy minimization). The Subgradient Push algorithm converges sublinearly at a rate of $\mathcal{O}\left(\ln k/\sqrt{k}\right)$ for general convex functions [57], and at a rate of $\mathcal{O}\left(\ln k/k\right)$ for strongly convex functions [55].

2.4 Extra Push

The first order state-of-the-art Extra-Push optimization algorithm [88] is an extension of the original doubly-stochastic EXTRA (exact first order algorithm) [70] to singly-stochastic operation. Most notably, agents running the Extra-Push algorithm achieve consensus and converge to a global minimum by using a constant step-size. It is this ability to use a constant step-size, rather than a diminishing one, that attributes Extra-Push with its linear convergence rate. At all time-indices k, each agent v_i locally maintains the variables $w_i[k], z_i[k] \in \mathbb{R}^d$, and $y_i[k] \in \mathbb{R}$. In order to describe the algorithm from a global perspective, we use the matrix-based formulation of the synchronous Extra-Push method provided in Algorithm 4. The consensus-matrices $\boldsymbol{P}_0$ are as defined in (2.4). The matrix-valued function $\nabla \boldsymbol{F}[k]$ is the Jacobian of the global objective $F(\cdot)$ evaluated at $\boldsymbol{z}[k]$. In the non-differentiable, but convex objective setting, the rows of $\nabla \boldsymbol{F}[k]$ are members of the subgradient set of the global objective $F(\cdot)$ evaluated at $\boldsymbol{z}[k]$. Since the algorithm's updates

Algorithm 4 Synchronous Extra-Push (cf. [88])

for $k = 2, 3, 4, \ldots$ **to** termination **do**

$$\boldsymbol{w}[k+1] = (\boldsymbol{P}_0 + \mathbf{I}_n)\,\boldsymbol{w}[k] - \frac{1}{2}(\boldsymbol{P}_0 + \mathbf{I}_n)\,\boldsymbol{w}[k-1] - \alpha\,(\nabla \boldsymbol{F}[k] - \nabla \boldsymbol{F}[k-1]) \tag{2.8}$$

$$y[k+1] = \boldsymbol{P}_0 y[k] \tag{2.9}$$

$$\boldsymbol{z}[k+1] = \mathrm{diag}(y[k+1])^{-1}\boldsymbol{w}[k+1] \tag{2.10}$$

requires iterates from two iterations back, we must provide initializations for $k = 0$ and $k = 1$. The initializations at time-index $k = 0$ are $\boldsymbol{z}[0] = \boldsymbol{w}[0] \in \mathbb{R}^d$, $y[0] = \mathbf{1}_n$. The initializations at time-index $k = 1$ are $\boldsymbol{w}[1] = \boldsymbol{P}_0\boldsymbol{w}[0] - \alpha\nabla \boldsymbol{F}[0]$, $\boldsymbol{y}[1] = \boldsymbol{P}_0\boldsymbol{y}[0]$, and $\boldsymbol{z}[1] = \mathrm{diag}(y[1])^{-1}w[1]$. The constant α is just a positive scalar step-size. The Extra-Push algorithm can be directly derived from the synchronous Subgradient-Push algorithm by taking the difference between two successive iterations of (2.5); one iteration using the consensus-matrix $\boldsymbol{P}_0$ and the other using the consensus-matrix $\widetilde{\boldsymbol{P}}_{\mathbf{0}} := \frac{1}{2}(\mathbf{I}_n + \boldsymbol{P}_0)$. By using a gradient difference in the optimization update, Extra-Push is able to achieve exact convergence for general convex functions using a constant step-size. The literature on the convergence theory for the Extra-Push algorithm is restricted to synchronous and static (time-invariant) directed graphs. The Extra-Push algorithm converges Q-linearly when the global objective is strongly convex [58, 88].

2.5 Push DIGing

The Push-DIGing (distributed gradient tracking) optimization algorithm [58] is another first order state-of-the-art method. Most notably, agents running the Push-DIGing algorithm achieve consensus and converge to a global minimum, even

Algorithm 5 Synchronous Push-DIGing (cf. [58])

for $k = 0, 1, 2, \ldots$ **to** termination **do**

$$\boldsymbol{w}[k+1] = \boldsymbol{P}_0 \left(\boldsymbol{w}[k] - \alpha \boldsymbol{x}[k]\right) \tag{2.11}$$
$$y[k+1] = \boldsymbol{P}_0 y[k] \tag{2.12}$$
$$\boldsymbol{z}[k+1] = \operatorname{diag}(y[k+1])^{-1} \boldsymbol{w}[k+1] \tag{2.13}$$
$$\boldsymbol{x}[k+1] = \boldsymbol{P}_0 \boldsymbol{x}[k] + (\nabla \boldsymbol{F}[k+1] - \nabla \boldsymbol{F}[k]) \tag{2.14}$$

in the presence of time-varying communication topologies, by using a constant step-size. It is this ability to use a constant step-size, rather than a diminishing one, that attributes Push-DIGing with its linear convergence rate, and it is the dynamic gradient tracking procedure that allows the method to converge over time-varying graphs (in contrast to Extra-Push which diverges over time-varying graphs [59]). At all time-indices k, each agent v_i locally maintains the variables $w_i[k], x_i[k], z_i[k] \in \mathbb{R}^d$, and $y_i[k] \in \mathbb{R}$. In order to describe the algorithm from a global perspective, we use the matrix-based formulation of the synchronous Push-DIGing method provided in Algorithm 5.

The consensus-matrices $\boldsymbol{P}_0$ are as defined in (2.4). The matrix-valued function $\nabla \boldsymbol{F}[k]$ is the Jacobian of the global objective $F(\cdot)$ evaluated at $\boldsymbol{z}[k]$. In the non-differentiable, but convex objective setting, the rows of $\nabla \boldsymbol{F}[k]$ are members of the subgradient set of the global objective $F(\cdot)$ evaluated at $\boldsymbol{z}[k]$. The initializations are $\boldsymbol{z}[0] = \boldsymbol{w}[0] \in \mathbb{R}^d$, $y[0] = \mathbf{1}_n$, $\boldsymbol{x}[0] = \nabla \boldsymbol{F}[0]$ and α is just a positive scalar step-size. The algorithm shares many similarities to Extra-Push, however one of the most salient differences is that the Push-DIGing method performs dynamic gradient tracking through the variable x, and is therefore capable of converging over time varying

directed graphs. The synchronous Push-Diging algorithm converges R-linearly when the global objective is strongly convex [58].

2.6 Summary

In this chapter we reviewed the Push-Sum consensus-averaging algorithm — a fundamental building block of singly-stochastic multi-agent optimization — and provided a brief intertextual exposition of some of the state-of-the-art methods in multi-agent optimization. All of these methods have only been described and analyzed in the synchronous setting in the literature. In this thesis we analyze an asynchronous version of Subgradient-Push, and, to that end, an asynchronous version of the Push-Sum consensus-averaging algorithm. In the next chapter we discuss the system model that we introduce to facilitate this analysis.

CHAPTER 3
System Model

To describe Asynchronous Gradient Push, and prove convergence, we establish some new notation, adapt existing graph-based communication models, and slightly update the optimization iteration of Synchronous Subgradient Push. We put together these pieces to create an asynchronous multi-agent model under the assumptions of bounded computation and communication delays. Without any loss of generality we can describe and analyze the algorithm in discrete-time since all events of interest, such as message transmissions/receptions and local variable updates, may be indexed by a discrete-time variable (cf. [78]).

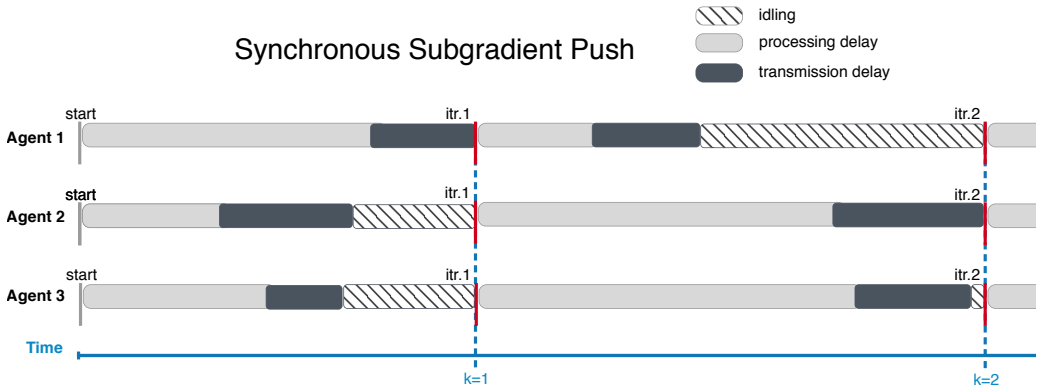
For analysis purposes we adopt the notation and terminology for analyzing asynchronous algorithms developed in [78, 5]. We let $t[0]$ denote the time at which the agents begin optimization, and we assume that there is a set of times $T = \{t[1], t[2], t[3], \dots, \}$ at which one or more agents perform an update. We let $T_i \subseteq T$ denote the subset of times at which agent v_i in particular performs an update. For example, if the *time-indices* at which agent v_i performs an update are given by $\{5, 6, 25, \dots, \}$, then $T_i = \{t[5], t[6], t[25] \dots, \}$. In this case, $t[5]$ is the time at which agent v_i completes its first **Local Computation**, $t[6]$ is the time at which agent v_i completes its second **Local Computation**, $t[25]$ is the the time at which agent v_i completes its third **Local Computation**, and so on. Since agents do not wait for each other to complete computations, nor do they wait for messages to be received

before moving on to the next step in the algorithm, it follows that the differences $t[5]-t[0]$, $t[6]-t[5]$, $t[25]-t[6]$ can be regarded as the inter-update delays; we refer to these time differences as the *continuous-time processing delays* experienced by agent v_i, and we refer to the corresponding time-index differences, $5-0$, $6-5$, $25-6$, etc., as the *time-index processing delays* experienced by agent v_i. Assume that the continuous-time processing delays are bounded from above and below — meaning that agents do not take an infinite amount of time to perform an update, nor do they perform updates infinitely fast — as is typically the case in works that analyze asynchronous algorithms (cf. Bertsekas and Tsitsiklis [78, 5]), then it follows that the time-index processing delays are also bounded. Let $\overline{\tau}^{\text{proc}}$ denote an upper bound on the time-index processing delays.

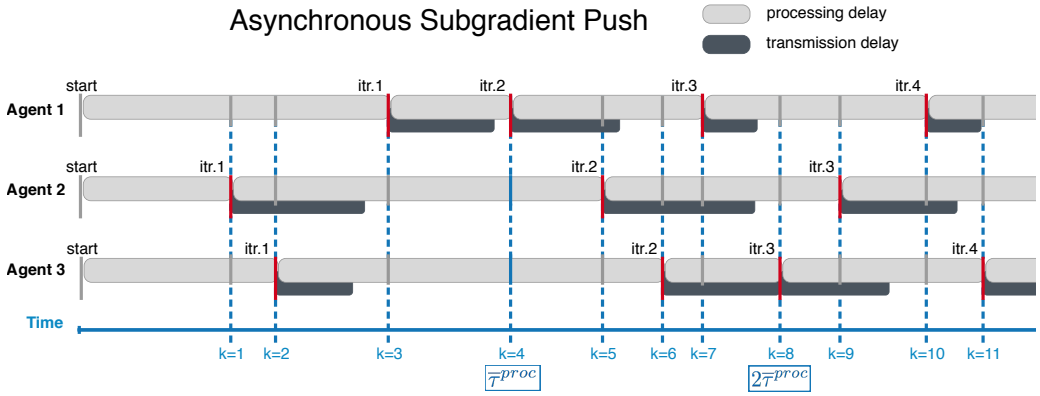
In practical implementations of message-passing algorithms, when a message is sent from one agent to another, that message experiences some real (non-integer valued) transmission delay due to the variable latency in the communication medium. Since agents neither wait for each other to complete computations, nor do they wait for messages to be sent/received before moving on to their next local iteration, it follows that an outgoing message may not be processed until some other time later in the future. For example, a message sent from agent v_i to agents v_j and v_ℓ at time $t[k]$ may experience some variable transmission delays $\tau_{j'}, \tau_{\ell'} \in \mathbb{R}_+$. Thus, the messages may not be processed by agents v_j and v_ℓ until some later times $t[j'] \geq t[k] + \tau_{j'}$ and $t[\ell'] \geq t[k] + \tau_{\ell'}$ respectively. The fact that the message arrives at agent v_j's receive-buffer at time $t[k] + \tau_{j'}$ does not necessarily imply that it is also processed at time $t[k]+\tau_{j'}$; it may be that agent v_j is busy performing a **Local Computation**, and does

not process the newly arrived message until its next update. The continuous-time differences $t[j'] - t[k]$ and $t[\ell'] - t[k]$ are referred to as the *continuous-time message delays*, and the corresponding time-index differences $k' - k$ and $s' - k$ are referred to as the *time-index message delays*. Assume that the continuous-time message delays are bounded, as is typically the case in works that analyze asynchronous algorithms (cf. Bertsekas and Tsitsiklis [78, 5]), then, coupled with the assumption that continuous-time processing delays are bounded from above and below, it follows that the time-index message delays are also bounded. Let $\overline{\tau}^{\text{msg}}$ denote an upper bound on the time-index message delays.

Note that we have used $t[k]$ to refer to the time at which an agent completes a **Local Computation** — performs an update — and also the time at which that same agent begins **Asynchronous Gossip** — sends a message to its neighbours by copying the outgoing message into its local send-buffer. And so we say that an agent performs an update and sends a message to its out-neighbours at time $t[k]$. More precisely, $t[k] \in T_i$ implies that agent v_i performs an update and sends a message to its out-neighbours at time $t[k]$. Furthermore, since messages are only processed during the agent update times, it follows that, for analysis purposes, messages are sent with an *effective delay* such that they arrive right when the agent is ready to process the messages. That is, a message sent at time $t[k]$ that arrives in an agent's receive buffer at time $t[k] + \tau_{j'}$, but is not processed until time $t[j']$, is, for analysis purposes, sent with a continuous-time delay $t[j'] - t[k]$, or equivalently a time-index delay $j' - k$. Figure 3–1a illustrates the agent update procedure in the synchronous



(a)



(b)

Figure 3–1: Example of agent updates in synchronous and asynchronous Subgradient Push implementations with $\overline{\tau}^{\text{proc}} = 4$ in the asynchronous case. Processing delays correspond to the time required to perform a local iteration. Transmission delays correspond to the time required for all outgoing message to arrive at their destination buffers. Even though a message arrives at a destination agent's receive-buffer after some real (non-integer valued) delay, that message is only processed when the destination agents performs its next update.

case (Synchronous Subgradient Push): agents must wait for all network communications to be complete before moving-on to the next iteration, and, as a result, some agents may experience idling periods. Figure 3–1b, on the other hand, illustrates the agent update procedure in the asynchronous case (Asynchronous Subgradient

Push): at the end of each local iteration, agents make use of their message buffers by copying all outgoing messages into their local send-buffers, and by retrieving all messages from their local receive-buffers. The underlying communication systems subsequently transmit the messages in the send-buffers while the agents proceed with their computations. Though not totally obvious from the diagram, it *is* possible for message transmission delays to take more time than processing delays; this is a non-issue so long as the agents' communication buffers do not overflow in practice.

3.1 Communication

The multi-agent communication topology is represented by a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \coloneqq \{v_i \mid i = 1, \ldots, n\}$ is the set of agents, and $\mathcal{E} \coloneqq \{(v_j \leftarrow v_i) \mid v_i \text{ can send messages to } v_j\}$ is the set of edges. We refer to $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as the *reference graph* for reasons that will become apparent when we augment the graph with virtual agents. We let $N_j^{\text{in}} \coloneqq |\{v_i \mid (v_j \leftarrow v_i) \in \mathcal{E}\}|$ denote the cardinality of the *in-neighbour* set of agent v_j, and $N_j^{\text{out}} \coloneqq |\{v_i \mid (v_i \leftarrow v_j) \in \mathcal{E}\}|$ denote the cardinality of the *out-neighbour* set of agent v_j. Since all agent have access to their local information at every iteration, we use the convention that each agent is both an in- and out-neighbour of itself at all times; *i.e.*, $(v_i \leftarrow v_i) \in \mathcal{E}$ for all i. We define the *communication set* at time-index k by $\mathcal{C}[k]$, and we say that $v_i \in \mathcal{C}[k]$ if and only if agent v_i performs an update at time $t[k]$. More formally, $\mathcal{C}[k] \coloneqq \{v_i \mid t[k] \in T_i\}$. For convenience, we also define the functions $\pi_i(k) \coloneqq \max\{k' \in \mathbb{N} \mid k' < k,\ v_i \in \mathcal{C}[k']\}$ for all i, which return the most recent time-index — up to, but not including, time-index k — that agent v_i was in the communication set. To handle the corner-case at $k = 1$, we let $\pi_i(1)$ equal 0 for all i.

3.2 Delays

Let $\tau_i^{\text{proc}}[k] \coloneqq k - \pi_i(k)$ (defined for all $t[k] \in T_i$) denote the time-index processing delay experienced by agent v_i at time $t[k]$. Since the maximum time-index processing delay experienced by an agent is denoted $\overline{\tau}^{\text{proc}}$, we have that $1 \leq \tau_i^{\text{proc}}[k] \leq \overline{\tau}^{\text{proc}}$. In words, if agent v_i performs an update at some time $t[k]$, then it performed its last update at time $t[k - \tau_i^{\text{proc}}[k]]$, where $\tau_i^{\text{proc}}[k]$ is bounded above by $\overline{\tau}^{\text{proc}}$. Note that this is less strict than explicitly requiring the inter-update time $t[k] - t[k - \tau_i^{\text{proc}}[k]]$ to be bounded from above and below.

By similar construction, let $\tau_{ji}^{\text{msg}}[k]$ (defined for all $t[k] \in T_i$) denote the time-index message delay experienced by a message sent from agent v_i to agent v_j at time $t[k]$. Since the maximum time-index message delay is denoted $\overline{\tau}^{\text{msg}}$, we have that $0 \leq \tau_{ji}^{\text{msg}}[k] \leq \overline{\tau}^{\text{msg}}$. In words, if agent v_i sends a message to agent v_j at time $t[k]$, then agent v_j will process that message at time $t[k + \tau_{ji}^{\text{msg}}[k]]$, where $\tau_{ji}^{\text{msg}}[k]$ is bounded above by $\overline{\tau}^{\text{msg}}$. Note that this is less strict than explicitly requiring $t[k + \tau_{ji}^{\text{msg}}[k]]$ to be bounded. We use the convention that $\tau_{ii}^{\text{msg}}[k] = 0$ for all i and $k \in \mathbb{N}$, meaning that all agents have access to their own local information at all times.

Since all agents enter the communication set — *i.e.*, complete an update and initiate a message transmission to all their out-neighbours — at least once every $\overline{\tau}^{\text{proc}} - 1$ time indices, and because all messages are processed within at most $\overline{\tau}^{\text{msg}}$ time indices from when they are sent, it follows that each agent is guaranteed to process at least one message from each its in-neighbours every $\overline{\tau} \coloneqq \overline{\tau}^{\text{msg}} + \overline{\tau}^{\text{proc}} - 1$ time indices. In our subsequent analysis, we use $\overline{\tau}$ to derive the *effective connectivity* of the communication graph.

3.3 Augmented Graph

To analyze the Asynchronous Gradient Push optimization algorithm we augment the reference graph by adding $\overline{\tau}^{\text{msg}}$ virtual agents for each non-virtual agent, using a procedure similar to that used in [15, 16, 29, 30] for synchronous averaging with transmission delays. To state the procedure concisely: for each non-virtual agent, v_j, we add $\overline{\tau}^{\text{msg}}$ virtual agents, $v_j^{(1)}, v_j^{(2)}, \ldots, v_j^{(\overline{\tau}^{\text{msg}})}$, where each $v_j^{(r)}$ contains the messages to be received by agent v_j in r time indices. As an aside, we may interchangeably refer to the non-virtual agents, v_j, as $v_j^{(0)}$ for the purpose of notational consistency. The virtual agents associated with agent v_j are daisy-chained together with communication edges $(v_j^{(r-1)} \leftarrow v_j^{(r)})$, such that at each time-index k, and for all $r \in \{1, \ldots, \overline{\tau}^{\text{msg}}\}$, agent $v_j^{(r)}$ forwards its summed messages to agent $v_j^{(r-1)}$. In addition, for each edge $(v_j^{(0)} \leftarrow v_i^{(0)})$ in the reference graph (where $j \neq i$), we add the edges $(v_j^{(r)} \leftarrow v_i^{(0)})$ in the augmented graph.

This augmented model simplifies the subsequent analysis by enabling agent v_i to send a message at time-index k to agent $v_j^{(\tau_{ji}^{\text{msg}}[k])}$ with delay zero, rather than send a message to agent v_j with delay $\tau_{ji}^{\text{msg}}[k]$. An example of the graph augmentation procedure is shown in Figure 3–2. The solid agents and edges correspond to the reference graph, and the dashed agents and edges correspond to the those inserted after the graph augmentation. At this point, it is worth pointing out that we have not changed our definitions for the edge and vertex sets $\mathcal{E}$ and $\mathcal{V}$ respectively, they are still solely defined in-terms of the non-virtual agents.

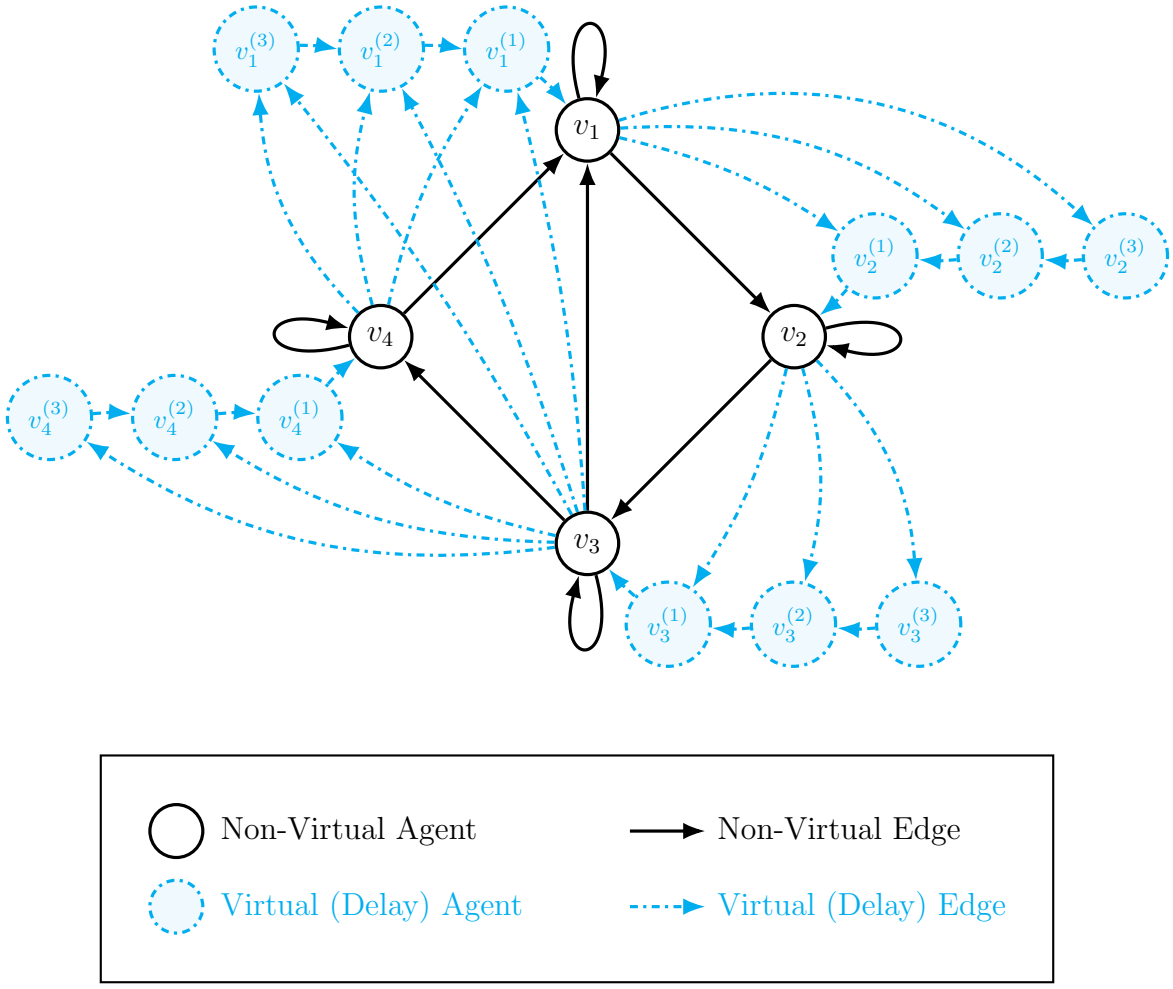


Figure 3–2: Sample augmented graph of a 4-agent reference network with a maximum time-index message transmission delay of $\overline{\tau}^{\text{msg}} = 3$ iterations.

To adapt the augmented graph model for optimization we formulate the equivalent optimization problem

$$\text{minimize} \quad \overline{F}(x) \coloneqq \sum_{r=0}^{\overline{\tau}^{\text{msg}}} \sum_{i=1}^{n} f_i^{(r)}(x), \tag{3.1}$$

where $f_i^{(r)}(x)$ equals $f_i(x)$ if $r = 0$, and $f_i^{(r)}(x)$ equals 0 if $r \neq 0$. In words, each of the non-virtual agents, $v_i^{(0)}$, maintains its original objective function $f_i(\cdot)$, and all the virtual agents are simply given the zero objective. Clearly $\overline{F}(x)$ defined in (3.1) is equal to $F(x)$ defined in (1.1), and we will use these two notations interchangeably.

We also define the augmented state matrix $\boldsymbol{x}[k] \in \mathbb{R}^{n(\overline{\tau}^{\mathrm{msg}}+1)\times d}$, given by

$$\boldsymbol{x}[k] \coloneqq \begin{bmatrix} \boldsymbol{x}^{(0)}[k] \\ \boldsymbol{x}^{(1)}[k] \\ \vdots \\ \boldsymbol{x}^{(\overline{\tau}^{\mathrm{msg}})}[k] \end{bmatrix},$$

where each $\boldsymbol{x}^{(r)}[k] \in \mathbb{R}^{n\times d}$ is a block matrix that stores a copy of the variable x at all the delay-r agents in the augmented graph at time-index k (in keeping with this notation, the block matrix $\boldsymbol{x}^{(0)}[k]$ corresponds to the non-virtual agents in the network). More specifically, $x_i^{(r)}[k] \in \mathbb{R}^d$, the i^{th} row of $\boldsymbol{x}^{(r)}[k]$, is a copy of the variable x held locally at agent $v_i^{(r)}$ at time-index k; we generalize this notation for other variables as well.

3.4 Brief Recap

We now briefly recap the behaviour of the asynchronous algorithm model. When a non-virtual agent is in the communication set (just completed and update), the agent asynchronously sends a message to each out-neighbour with some bounded delay, such the messages are received right when the destination agents are ready to process them, and, commensurately, reads all the messages that were received (buffered) while carrying out the previous local update. When a non-virtual agent is *not* in the communication set, the agent remains computing, neither sending new

messages to, nor receiving new messages from, any of its neighbours. Each non-virtual agent is guaranteed to return the communication set (preform an update) at least once every $\overline{\tau}^{\text{proc}}$ time indices, and no message experiences a time-index delay greater than $\overline{\tau}^{\text{msg}}$. The virtual agents simply forward all of their messages to the next agent in the delay daisy-chain at each subsequent time-index, and since all the virtual agents have $f_i^{(r)}(\cdot) \coloneqq 0$, they never produce any new information in their gradient step iterations (5.4). Hence, the non-virtual agents do not influence the result of the optimization. This model and formulation allows agents to work at independent rates, communicate with arbitrary transmission delays, and perform computations with outdated information, so long as the delays are bounded.

CHAPTER 4
Asynchronous Consensus using Perturbed Push Sum

Just as consensus-averaging is a fundamental building block of the synchronous state-of-the-art multi-agent optimization methods [72], so too is consensus-averaging a fundamental building block of the proposed Asynchronous Subgradient Push optimization algorithm. In this chapter we present and prove convergence of an asynchronous version of the synchronous Perturbed Push-Sum Protocol [57], the goal of which is to perform push-sum consensus averaging with some unknown perturbation term (*e.g.*, random noise or a local gradient) added to the agents' iterates at each local iteration. Our definition of asynchrony in the context of push-sum consensus implies that agents may gossip with their neighbours at different rates (due to heterogeneous/variable message-processing delays), communicate with arbitrary transmission delays, and perform push-sum averaging steps with stale (outdated) information. This behaviour is in stark contrast to that of the synchronous Push-Sum consensus-averaging algorithm defined in Section 2.2, where all agents gossip at the same rate, and idle at each communication round until all messages to/from neighbours are sent/received.

4.1 Formulation of Asynchronous Push-Sum

The lack of synchronization between agents in the asynchronous Perturbed Push-Sum formulation results in specific times at which some agents are communicating, and others are not, leading to an effectively time-varying graph structure, $\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k])$,

which we appropriately call the *effective graph*. Note the subtle distinction between $\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k])$ used to represent the time-varying effective graph, and $\mathcal{G}(\mathcal{V}, \mathcal{E})$ used to represent the static reference graph. The effective graph is a subgraph of the reference graph (not the augmented graph) with the same exact vertex-set, but only a subset of the edge-set at each time-index. For example, if agent v_i initiates a message transmission to agent v_j at time $t[k]$, then the edge $(v_j \leftarrow v_i)$ is in the effective graph at time $t[k]$, $\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k])$, even though that message experiences some finite-valued transmission delay, and may not be processed by agent v_j until some later time $t[k']$. The effective graph, $\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k])$, is used solely for analysis purposes to denote the set of edges in the reference graph along which message transmissions are initiated at time $t[k]$. Since the non-virtual agents are guaranteed to send a message to each of their peers at least once every $\overline{\tau}^{\text{proc}}$ time indices, it follows that the union of $\overline{\tau}^{\text{proc}}$ consecutive effective graphs is equivalent to the reference graph in terms of the non-virtual agent connectivity (*i.e.*, $\bigcup_{t=0}^{\overline{\tau}^{\text{proc}}-1} \mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k+t]) \equiv \mathcal{G}(\mathcal{V}, \mathcal{E})$ for all $k \geq 0$). Therefore, if the reference graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, is strongly connected, then the effective graph sequence $\{\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k])\}$ is $\overline{\tau}^{\text{proc}}$-strongly connected.

The consensus matrices $\overline{\boldsymbol{P}}[k] \in \mathbb{R}^{n(\overline{\tau}^{\mathrm{msg}}+1)\times n(\overline{\tau}^{\mathrm{msg}}+1)}$ for the augmented state model are defined as

$$\overline{\boldsymbol{P}}[k] \coloneqq \begin{bmatrix} \widetilde{\boldsymbol{P}}_0[k] & \boldsymbol{I}_{n\times n} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \widetilde{\boldsymbol{P}}_1[k] & \boldsymbol{0} & \boldsymbol{I}_{n\times n} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \widetilde{\boldsymbol{P}}_{\overline{\tau}-1}[k] & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{I}_{n\times n} \\ \widetilde{\boldsymbol{P}}_{\overline{\tau}^{\mathrm{msg}}}[k] & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \end{bmatrix}, \tag{4.1}$$

where each $\widetilde{\boldsymbol{P}}_r[k] \in \mathbb{R}^{n\times n}$ is a block matrix defined as

$$\left[\widetilde{\boldsymbol{P}}_r[k]\right]_{ji} \coloneqq \begin{cases} \frac{1}{N_i^{\mathrm{out}}}, & v_i \in \mathcal{C}[k],\ (j,i) \in \mathcal{E},\ \text{and } \tau_{ji}^{\mathrm{msg}}[k] = r, \\ 1, & v_i \notin \mathcal{C}[k],\ r = 0,\ j = i, \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

In words, when a non-virtual agent is in the communication set, it sends a message to each of its out-neighbours in the reference graph with some arbitrary, but bounded, delay r, where the delay r may vary from one out-neighbour to another, and from one time instance to another. When a non-virtual agent is not in the communication set, it keeps its value and does not gossip. Furthermore, since we have chosen a convention in which messages between agents are sent with some effective message delay, $\tau_{ji}^{\mathrm{msg}}[k]$, it follows than non-virtual agents do not process any messages while outside the communication set. Virtual agents, on the other hand, simply forward all of their messages to the next agent in the delay daisy-chain at all time-indices k, and so there is no notion of virtual agents belonging to (or not belonging to) the

Algorithm 6 Asynchronous Perturbed Push-Sum Averaging

for $k = 0, 1, 2, \ldots$ **to** termination **do**

$$\boldsymbol{w}[k+1] = \overline{\boldsymbol{P}}[k]\boldsymbol{x}[k] \tag{4.3}$$
$$y[k+1] = \overline{\boldsymbol{P}}[k]y[k] \tag{4.4}$$
$$\boldsymbol{z}[k+1] = \mathrm{diag}(y[k+1])^{-1}\boldsymbol{w}[k+1] \tag{4.5}$$
$$\boldsymbol{x}[k+1] = \boldsymbol{w}[k+1] + \boldsymbol{\eta}[k+1] \tag{4.6}$$

communication set. The communication set is exclusively a construct for the non-virtual agents. In the sequel, we make frequent use of the block matrix definitions $\widetilde{\boldsymbol{P}}_r[k]$ defined in (4.2).

Remark 1 (Properties of Consensus Matrices)**.** *Observe that the matrices* $\overline{\boldsymbol{P}}[k]$ *are column stochastic at all time-indices* k *simply by their definition. Furthermore, the block matrix sum* $\sum_{r=0}^{\overline{\tau}^{msg}} \widetilde{\boldsymbol{P}}_r[k']$ *conforms to the* effective graph *representing the non-virtual agent connectivity at time* k'*:* $\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k'])$*. Since the union of* $\overline{\tau}^{proc}$ *consecutive effective graphs is equivalent to the reference graph in terms of the non-virtual agent connectivity (i.e.,* $\cup_{k'=0}^{\overline{\tau}^{proc}} \overline{\mathcal{E}}[k+k'] \equiv \mathcal{E}$*), we have that the block matrix sum* $\sum_{k'=1}^{\overline{\tau}^{proc}} \sum_{r=0}^{\overline{\tau}^{msg}} \widetilde{\boldsymbol{P}}_r[k+k']$ *(for any* $k \in \mathbb{N}$*) conforms to the reference graph structure* $\mathcal{G}(\mathcal{V}, \mathcal{E})$*.*

To analyze the Asynchronous Perturbed Push-Sum Averaging algorithm from a global perspective, we us the matrix-based formulation provided in Algorithm 6, where $\boldsymbol{\eta}[k+1] \in \mathbb{R}^{n(\overline{\tau}^{\mathrm{msg}}+1)\times d}$ is some perturbation term, and the matrices $\overline{\boldsymbol{P}}[k]$ are as defined in (4.1) for the augmented state. At all time-indices k, each agent $v_i^{(r)}$ locally maintains the variables $w_i^{(r)}[k]$, $z_i^{(r)}[k]$, $x_i^{(r)}[k] \in \mathbb{R}^d$, and $y_i^{(r)}[k] \in \mathbb{R}$. This matrix-based formulation describes how the agents' values evolve at some time

$t[k+1] \in T = \{t[1], t[2], t[3], \ldots, \}$ — a time at which one or more agents complete an update, which in this case consists of processing (summing) received messages. The time-varying consensus-matrices $\overline{\boldsymbol{P}}[\cdot]$ capture the asynchronous communication dynamics between agents.

4.2 Main Results

Assumption 1 (Communicability)**.** *All agents influence each other's values sufficiently often; precisely:*

1. *The reference graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is static and strongly connected.*
2. *The communication and computation delays are bounded: $\overline{\tau}^{msg} < \infty$ and $\overline{\tau}^{proc} < \infty$.*

Remark 2. *The assumption that the reference graph is static is only made for ease of exposition. In the discussion section we explain how one can extend the analysis to account for time-varying directed communication topologies.*

Let n represent the number of non-virtual agents in the network; let the scalar ψ represents the number of possible types (zero/non-zero structures) that an $n \times n$ SIA (Stochastic, Indecomposable, and Aperiodic) matrix can take (hence $\psi < 2^{n^2}$); let the scalar λ represent the maximum Hajnal and Bartlett Coefficient of Ergodicity (cf. [31]) taken over the product of all possible $(\overline{\tau}+1)$ consensus-matrix products — we will prove later that λ is strictly less than 1, and guaranteed to exist — and let δ_{min} represent a lower bound on the entries in the first n-rows of the product of $n(\overline{\tau}+1)$ or more consecutive consensus-matrices (rows corresponding to the non-virtual agents).

Theorem 1 (Convergence Rate of Asynchronous Perturbed Push-Sum Averaging)**.** *Suppose that Assumption 1 is satisfied, then it holds for all $i = 1, 2, \ldots, n$, and $k \geq 0$,*

that

$$\left\| z_i^{(0)}[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} \right\|_1 \leq C q^k \left\| x_i^{(0)}[0] \right\|_1 + C \sum_{s=0}^{k} q^{k-s} \left\| \eta_i[s] \right\|_1 ,$$

where $q \in (0,1)$ *is related to the degree of ergodicity associated with the asymptotic product of the consensus-matrices,* $q = \lambda^{\frac{1}{(\psi+1)(\overline{\tau}+1)}}$*, and* C *is a finite constant,* $C < \frac{2}{\lambda^{(\psi+2)/(\psi+1)}\delta_{min}} \approx \frac{2}{\lambda \delta_{min}}$ *with* $\delta_{min} = \min_{1 \leq j \leq n} \left(\frac{1}{N_j^{out}} \right)^{n(\overline{\tau}+1)}$.

Remark. *Note that Theorem 1 is only stated with regards to the consensus estimates of the non-virtual agents:* $\left\{ z_i^{(0)} \in \mathbb{R}^d \;\middle|\; i = 1, 2, \ldots, n \right\}$*. As for the virtual agents, the constant* C *may blow-up and result in a trivial bound since the corresponding entries in the consensus matrices may vanish (equal 0). Also note that while* C *appears to have an inverse proportionality to the second largest eigenvalue of the graph consensus matrices,* λ*, the constant* q *exhibits a stronger exponential relationship to* λ*, and so smaller values of* λ *greatly improve the convergence rate bound. Furthermore, observe that both* q *and* C *depend on the maximum effective delay in the network; specifically, larger delays will result in larger values of both* q *and* C*, meaning that larger effective delays result in slower convergence bounds. Lastly, it should be noted that the scalar* ψ*, which represents the number of possible types (zero/non-zero structures) that an* $n \times n$ *SIA matrix can take, increases with the number of agents* n*; therefore, increasing the number of agents results in a slower convergence rate bound. In general, the bound provided on* ψ*,* $(\leq 2^{n^2})$ *is very loose and can possibly be tightened by taking into account the graph conformance and SIA properties of the consensus matrices.*

Remark. *It is worth pointing out that the result of Theorem 1 is interesting in that it informs us on the asymptotic convergence of the Asynchronous Perturbed Push-Sum Averaging algorithm and allows us to couple this method with distributed optimization*

algorithms (the geometric rate of convergence is typically crucial to the analysis of overlying distributed optimization methods); however, the magnitude of the constant C is likely to be very large, and so the "bound" on performance is likely to be very loose and of little interest in practice.

Corollary 1.1 (Convergence to a Neighbourhood for Non-Diminishing Perturbation)**.** *If the perturbation term is bounded for all $i = 1, 2, \ldots, n$: there exists a constant $L < \infty$ such that*

$$\|\eta_i[k]\|_1 \leq L,$$

then by substituting the bound into the result of Theorem 1 and taking the limit of the geometric series, we have for all $i = 1, 2, \ldots, n$ that

$$\lim_{k\to\infty} \left\| z_i^{(0)}[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} \right\|_1 \leq \frac{\widetilde{C}L}{1-q}.$$

Remark 3. *From [67, Lemma 3.1] we know that if $q \in (0, 1)$, and $\lim_{s\to\infty} \alpha[s] = 0$, then it holds that*

$$\lim_{k\to\infty} \sum_{s=0}^{k} q^{k-s}\alpha[s] = 0.$$

Corollary 1.2 (Exact Convergence for Vanishing Perturbation)**.** *If the perturbation term tends to $\mathbf{0}$ as k (the time-index) tends to infinity,*

$$\lim_{k\to\infty} \|\boldsymbol{\eta}[\boldsymbol{k}]\|_1 = 0,$$

then from the result of Theorem 1 and Remark 3, it holds for all $i = 1, 2, \ldots, n$ that

$$\lim_{k\to\infty} \left\| z_i^{(0)}[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} \right\|_1 = 0.$$

4.3 Analysis

In order to prove that the consensus estimates of the non-virtual agents converge to a neighbourhood of the mutual time-wise average at a geometric rate, we show that the asymptotic product of the time-varying consensus-matrices, $\overline{\boldsymbol{P}}[k] \cdots \overline{\boldsymbol{P}}[1]\overline{\boldsymbol{P}}[0]$ (for large enough k) is stochastic, indecomposable, and aperiodic (SIA) and, furthermore, that the entries in the first n rows of the asymptotic product (corresponding to the non-virtual agents) are bounded below by a strictly positive quantity. Applying some standard tools from the literature concerning SIA matrices we show that the columns of the asymptotic product of consensus-matrices weakly converge to a (possibly time-varying) stochastic vector sequence at a geometric rate (*i.e.*, the columns of the asymptotic product all converge to one-another geometrically fast, but are not necessarily stationary). Substituting this geometric bound into the definition of the asynchronous perturbed Push-Sum updates in Algorithm 6, and, through a little algebraic manipulation, we obtain the desired result.

Lemma 1.1 (SIA Matrix Products)**.** *The product of $\ell + 1$ consecutive consensus-matrices $\boldsymbol{Q}_{\ell+1}[k] \coloneqq \overline{\boldsymbol{P}}[k+\ell] \ldots \overline{\boldsymbol{P}}[k+2]\overline{\boldsymbol{P}}[k]$ is (column) stochastic, indecomposable and aperiodic (SIA) for arbitrary k and $\ell \geq \overline{\tau}$.*

Proof. Column stochasticity of the product follows since each of the individual matrices $\overline{\boldsymbol{P}}[\cdot]$ are column stochastic by definition, and the product of column stochastic matrices is also column stochastic. Indecomposability and aperiodicity of the matrix product can be checked from the graph structure implied by the zero/non-zero structure of $\boldsymbol{Q}_{\ell+1}[k]$. In particular, a matrix is indecomposable if the directed graph it describes is connected and has only one strongly-connected component [30, 80].

An indecomposable matrix is aperiodic if it has at least one self-loop (this is a sufficient, but not a necessary, condition; cf. [30]). Since all of the non-virtual agents have self-loops at all times, aperiodicity will be satisfied by default. Therefore, what remains is to check indecomposability from the graph structure implied by the zero/non-zero structure of the matrix product. We represent the product $\boldsymbol{Q}_{\ell+1}[k] \coloneqq \overline{\boldsymbol{P}}[k+\ell]\dots\overline{\boldsymbol{P}}[k+2]\overline{\boldsymbol{P}}[k]$ as a block matrix

$$\boldsymbol{Q}_{\ell+1}[k] = \begin{bmatrix} \boldsymbol{Q}_{\ell+1}^{(0,0)} & \boldsymbol{Q}_{\ell+1}^{(0,1)} & \cdots & \boldsymbol{Q}_{\ell+1}^{(0,\overline{\tau}^{\mathrm{msg}})} \\ \boldsymbol{Q}_{\ell+1}^{(1,0)} & \boldsymbol{Q}_{\ell+1}^{(1,1)} & \cdots & \boldsymbol{Q}_{\ell+1}^{(1,\overline{\tau}^{\mathrm{msg}})} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{Q}_{\ell+1}^{(\overline{\tau}^{\mathrm{msg}},0)} & \boldsymbol{Q}_{\ell+1}^{(\overline{\tau}^{\mathrm{msg}},1)} & \cdots & \boldsymbol{Q}_{\ell+1}^{(\overline{\tau}^{\mathrm{msg}},\overline{\tau}^{\mathrm{msg}})} \end{bmatrix}.$$

We want to show that for $\ell \geq \overline{\tau}$ the $\boldsymbol{Q}_{\ell+1}^{(0,0)}$ matrix block conforms to a strongly connected graph, and that the remaining $\boldsymbol{Q}_{\ell+1}^{(0,1)}, \boldsymbol{Q}_{\ell+1}^{(0,2)}, \dots, \boldsymbol{Q}_{\ell+1}^{(0,\overline{\tau}^{\mathrm{msg}})}$ matrix blocks all have positive entries on their diagonal. The former condition implies that all the non-virtual agents form a strongly connected component, and the latter condition implies that all the virtual agents are connected to the strongly connected component. This is similar to the approach taken in [29] to prove the SIA property of the product of consensus matrices defined for an augmented graph.

Since $\boldsymbol{Q}_{\ell+1}[k]$ equals $\overline{\boldsymbol{P}}[k+\ell]\boldsymbol{Q}_\ell[k]$, we can rewrite $\boldsymbol{Q}_{\ell+1}$ in terms of the $\boldsymbol{Q}_\ell^{(i,j)}$ blocks

$$\boldsymbol{Q}_{\ell+1}[k] = \begin{bmatrix} \widetilde{\boldsymbol{P}}_0[k+\ell]\boldsymbol{Q}_\ell^{(0,0)} + \boldsymbol{Q}_\ell^{(1,0)} & \cdots & \widetilde{\boldsymbol{P}}_0[k+\ell]\boldsymbol{Q}_\ell^{(0,\overline{\tau}^{\mathrm{msg}})} + \boldsymbol{Q}_\ell^{(1,\overline{\tau}^{\mathrm{msg}})} \\ \widetilde{\boldsymbol{P}}_1[k+\ell]\boldsymbol{Q}_\ell^{(0,0)} + \boldsymbol{Q}_\ell^{(2,0)} & \cdots & \widetilde{\boldsymbol{P}}_0[k+\ell]\boldsymbol{Q}_\ell^{(0,\overline{\tau}^{\mathrm{msg}})} + \boldsymbol{Q}_\ell^{(2,\overline{\tau}^{\mathrm{msg}})} \\ \vdots & \cdots & \vdots \\ \widetilde{\boldsymbol{P}}_{\overline{\tau}^{\mathrm{msg}}-1}[k+\ell]\boldsymbol{Q}_\ell^{(0,0)} + \boldsymbol{Q}_\ell^{(\overline{\tau}^{\mathrm{msg}},0)} & \cdots & \widetilde{\boldsymbol{P}}_{\overline{\tau}^{\mathrm{msg}}-1}[k+\ell]\boldsymbol{Q}_\ell^{(0,\overline{\tau}^{\mathrm{msg}})} + \boldsymbol{Q}_\ell^{(\overline{\tau}^{\mathrm{msg}},\overline{\tau}^{\mathrm{msg}})} \\ \widetilde{\boldsymbol{P}}_{\overline{\tau}^{\mathrm{msg}}}[k+\ell]\boldsymbol{Q}_\ell^{(0,0)} & \cdots & \widetilde{\boldsymbol{P}}_{\overline{\tau}^{\mathrm{msg}}}[k+\ell]\boldsymbol{Q}_\ell^{(0,\overline{\tau}^{\mathrm{msg}})} \end{bmatrix},$$

from which we can see that

$$\boldsymbol{Q}_{\ell+1}^{(0,0)} = \widetilde{\boldsymbol{P}}_0[k+\ell]\boldsymbol{Q}_\ell^{(0,0)} + \boldsymbol{Q}_\ell^{(1,0)}.$$

Further decomposing the $\boldsymbol{Q}_{\ell+1}^{(0,0)}$ matrix block in terms of the $\boldsymbol{Q}_{\ell-1}^{(i,j)}$ matrix blocks gives

$$\boldsymbol{Q}_{\ell+1}^{(0,0)} = \widetilde{\boldsymbol{P}}_0[k+\ell]\left(\widetilde{\boldsymbol{P}}_0[k+\ell-1]\boldsymbol{Q}_{\ell-1}^{(0,0)} + \boldsymbol{Q}_{\ell-1}^{(1,0)}\right) + \widetilde{\boldsymbol{P}}_1[k+\ell-1]\boldsymbol{Q}_{\ell-1}^{(0,0)} + \boldsymbol{Q}_{\ell-1}^{(2,0)}. \quad (4.7)$$

After expanding the matrix product in the parentheses in (4.7) we have

$$\begin{aligned} \boldsymbol{Q}_{\ell+1}^{(0,0)} =& \widetilde{\boldsymbol{P}}_0[k+\ell]\widetilde{\boldsymbol{P}}_0[k+\ell-1]\boldsymbol{Q}_{\ell-1}^{(0,0)} \\ &+ \widetilde{\boldsymbol{P}}_0[k+\ell]\boldsymbol{Q}_{\ell-1}^{(1,0)} \\ &+ \boldsymbol{Q}_{\ell-1}^{(2,0)} \\ &+ \widetilde{\boldsymbol{P}}_1[k+\ell-1]\boldsymbol{Q}_{\ell-1}^{(0,0)}, \end{aligned}$$

which can be additionally decomposed in terms of the $\boldsymbol{Q}_{\ell-2}^{(i,j)}$ matrix blocks as

$$\begin{aligned}
\boldsymbol{Q}_{\ell+1}^{(0,0)} =& \widetilde{\boldsymbol{P}}_0[k+\ell]\widetilde{\boldsymbol{P}}_0[k+\ell-1]\widetilde{\boldsymbol{P}}_0[k+\ell-2]\boldsymbol{Q}_{\ell-2}^{(0,0)} \\
&+ \widetilde{\boldsymbol{P}}_0[k+\ell]\widetilde{\boldsymbol{P}}_0[k+\ell-1]\boldsymbol{Q}_{\ell-2}^{(1,0)} \\
&+ \widetilde{\boldsymbol{P}}_0[k+\ell]\boldsymbol{Q}_{\ell-2}^{(2,0)} \\
&+ \boldsymbol{Q}_{\ell-2}^{(3,0)} \\
&+ \widetilde{\boldsymbol{P}}_0[k+\ell]\widetilde{\boldsymbol{P}}_1[k+\ell-2]\boldsymbol{Q}_{\ell-2}^{(0,0)} \\
&+ \widetilde{\boldsymbol{P}}_1[k+\ell-1]\widetilde{\boldsymbol{P}}_0[k+\ell-2]\boldsymbol{Q}_{\ell-2}^{(0,0)} \\
&+ \widetilde{\boldsymbol{P}}_1[k+\ell-1]\boldsymbol{Q}_{\ell-2}^{(1,0)} \\
&+ \widetilde{\boldsymbol{P}}_2[k+\ell-2]\boldsymbol{Q}_{\ell-2}^{(0,0)}.
\end{aligned}$$

By recursing in a similar fashion, we have that the matrix block $\boldsymbol{Q}_{\ell+1}^{(0,0)}$ for $\ell \geq \overline{\tau}^{\mathrm{msg}}$ can be written as

$$\begin{aligned}
\boldsymbol{Q}_{\ell+1}^{(0,0)} =& \left(\Pi_{r=0}^{\ell-1}\widetilde{\boldsymbol{P}}_0[k+\ell-r]\right)\boldsymbol{Q}_1^{(0,0)} \\
&+ \left(\Pi_{r=0}^{\ell-2}\widetilde{\boldsymbol{P}}_0[k+\ell-r]\right)\boldsymbol{Q}_1^{(1,0)} \\
&\vdots \\
&+ \left(\Pi_{r=0}^{\ell-(\overline{\tau}^{\mathrm{msg}}+1)}\widetilde{\boldsymbol{P}}_0[k+\ell-r]\right)\boldsymbol{Q}_1^{(\overline{\tau}^{\mathrm{msg}},0)} \\
&+ \boldsymbol{D}_{\ell+1}^{(0,0)},
\end{aligned} \tag{4.8}$$

where $\boldsymbol{D}_{\ell+1}^{(0,0)}$ is some non-negative matrix. Substituting the definition for $\boldsymbol{Q}_1[k]$ ($\coloneqq \overline{\boldsymbol{P}}[k]$), we have

$$\begin{aligned}\boldsymbol{Q}_{\ell+1}^{(0,0)} &= \left(\Pi_{r=0}^{\ell-1}\widetilde{\boldsymbol{P}}_0[k+\ell-r]\right)\widetilde{\boldsymbol{P}}_0[k] \\ &\quad + \left(\Pi_{r=0}^{\ell-2}\widetilde{\boldsymbol{P}}_0[k+\ell-r]\right)\widetilde{\boldsymbol{P}}_1[k] \\ &\quad \vdots \\ &\quad + \left(\Pi_{r=0}^{\ell-(\overline{\tau}^{\mathrm{msg}}+1)}\widetilde{\boldsymbol{P}}_0[k+\ell-r]\right)\widetilde{\boldsymbol{P}}_{\overline{\tau}^{\mathrm{msg}}}[k] \\ &\quad + \boldsymbol{D}_{\ell+1}^{(0,0)},\end{aligned}$$

where the $\widetilde{\boldsymbol{P}}_r[k] \in \mathbb{R}^{n\times d}$ are the block matrices of $\overline{\boldsymbol{P}}[k]$ defined in (4.2) and (4.1) respectively.

Notice that if we left-multiply an arbitrary non-negative matrix $\boldsymbol{G}$ by an arbitrary non-negative matrix $\boldsymbol{F}$ that has all positive entries on its diagonals, the product $\boldsymbol{FG}$ will have positive entries in at least all the positions where $\boldsymbol{G}$ has positive entries. Since the matrix block $\widetilde{\boldsymbol{P}}_0[k]$, defined in (4.2), always has positive entries on its diagonals, from the convention that all non-virtual agents have self-loops at all time-indices k and that $\tau_{ji}^{\mathrm{msg}}[k] = 0$ whenever $j = i$, it follows that the matrix product coefficient terms $\Pi_{r=0}^{\ell-m}\widetilde{\boldsymbol{P}}_0[k+\ell-r]$ have positive entries on the diagonals. Therefore, it can be seen from (4.8) that the block matrix $\boldsymbol{Q}_{\ell+1}^{(0,0)}$ has positive entries in at least all the positions where $\sum_{r=0}^{\overline{\tau}^{\mathrm{msg}}}\widetilde{\boldsymbol{P}}_r[k]$ has positive entries. Using the fact that $\widetilde{\boldsymbol{P}}_0[\cdot]$ has positive entires on its diagonals, it follows that $\boldsymbol{Q}_{\ell+1}^{(0,0)}$ also has positive entries on its diagonals for all $\ell \geq 0$. Since we know from Remark 1 that the matrix sum $\sum_{r=0}^{\overline{\tau}^{\mathrm{msg}}}\widetilde{\boldsymbol{P}}_r[k]$ conforms to the graph structure of the effective graph at time-index k,

$\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k])$, it follows that the matrix block $\boldsymbol{Q}_{\ell+1}^{(0,0)}$, for all $\ell \geq \overline{\tau}^{\text{msg}}$, conforms to the graph structure of the effective graph at time-index k, $\mathcal{G}(\mathcal{V}, \overline{\mathcal{E}}[k])$.

Now consider two arbitrary non-negative matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ written in block matrix format as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{0,0} & \boldsymbol{A}_{0,1} & \dots & \boldsymbol{A}_{0,\overline{\tau}} \\ \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \dots & \boldsymbol{A}_{1,\overline{\tau}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}_{\overline{\tau},0} & \boldsymbol{A}_{\overline{\tau},1} & \dots & \boldsymbol{A}_{\overline{\tau},\overline{\tau}} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{0,0} & \boldsymbol{B}_{0,1} & \dots & \boldsymbol{B}_{0,\overline{\tau}} \\ \boldsymbol{B}_{1,0} & \boldsymbol{B}_{1,1} & \dots & \boldsymbol{B}_{1,\overline{\tau}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{B}_{\overline{\tau},0} & \boldsymbol{B}_{\overline{\tau},1} & \dots & \boldsymbol{B}_{\overline{\tau},\overline{\tau}} \end{bmatrix},$$

where the $\boldsymbol{A}_{0,0}$ and $\boldsymbol{B}_{0,0}$ blocks have strictly positive entries on their diagonal. The products $\boldsymbol{AB}$ and $\boldsymbol{BA}$ both have positive entries in the $(0,0)$ index block in at least all the position where $\boldsymbol{A}_{0,0}$ has positive entries.

Since $\widetilde{\boldsymbol{P}}_0[\cdot]$ (the (0,0) index block of $\overline{\boldsymbol{P}}[\cdot]$) has positive entries on its diagonals, it follows that left or right multiplying $\boldsymbol{Q}_{\ell+1}[k]$ by any consensus matrix $\overline{\boldsymbol{P}}[\cdot]$ (or a product thereof) will at least maintain all the positive entries in the $\boldsymbol{Q}_{\ell+1}^{(0,0)}$ block. Furthermore, since the product of $\overline{\tau}+1$ consecutive consensus-matrices matrices $\boldsymbol{Q}_{\overline{\tau}+1}[k]$ ($\coloneqq \overline{\boldsymbol{P}}[k+\overline{\tau}] \dots \overline{\boldsymbol{P}}[k+2]\overline{\boldsymbol{P}}[k]$) can be written as

$$\begin{aligned} \boldsymbol{Q}_{\overline{\tau}+1}[k] &= \boldsymbol{Q}_{\overline{\tau}^{\text{msg}}+1}[k+\overline{\tau}^{\text{proc}}-1]\overline{\boldsymbol{P}}[k+\overline{\tau}^{\text{proc}}-2]\dots\overline{\boldsymbol{P}}[k] \\ &= \overline{\boldsymbol{P}}[k+\overline{\tau}]\boldsymbol{Q}_{\overline{\tau}^{\text{msg}}+1}[k+\overline{\tau}^{\text{proc}}-2]\overline{\boldsymbol{P}}[k+\overline{\tau}^{\text{proc}}-3]\dots\overline{\boldsymbol{P}}[k] \\ &\vdots \\ &= \overline{\boldsymbol{P}}[k+\overline{\tau}]\overline{\boldsymbol{P}}[k+\overline{\tau}-1]\dots\overline{\boldsymbol{P}}[k+\overline{\tau}^{\text{msg}}+1]\boldsymbol{Q}_{\overline{\tau}^{\text{msg}}+1}[k], \end{aligned}$$

we have that the matrix block $\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}[k]$ has positive entries in at least all the same positions as the matrix $\boldsymbol{Q}^{(0,0)}_{\overline{\tau}^{\mathrm{msg}}+1}[k']$ for all $k' \in \{k, k+1, \ldots, k+\overline{\tau}^{\mathrm{proc}}-1\}$. Since each $\boldsymbol{Q}^{(0,0)}_{\overline{\tau}^{\mathrm{msg}}+1}[k']$ conforms to the effective graph structure at time-index k', we have that $\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}[k]$ conforms to the effective graph structure at $\overline{\tau}^{\mathrm{proc}}$ consecutive time-indices $(k, k+1, \ldots, k+\overline{\tau}-1)$. Recalling that the effective graph is $\overline{\tau}^{\mathrm{proc}}$-strongly connected (given our assumption that the reference graph is strongly connected [cf. Assumption 1]), it follows that the matrix block $\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}[k]$ conforms to a strongly connected graph component. Since the matrix block $\boldsymbol{Q}^{(0,0)}_{\ell+1}$, for all $\ell \geq \overline{\tau}$, has positive entries in at least all the same positions as the matrix block $\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}$, and because $\boldsymbol{Q}^{(0,0)}_{\ell+1}[k]$ has positive entries on its diagonals for all $\ell \geq 0$, it follows that $\boldsymbol{Q}^{(0,0)}_{\ell+1}[k]$ conforms to the (sub)graph structure of a strongly connected component and has positive entries on its diagonals for all $\ell \geq \overline{\tau}$.

Now it remains to be shown that the $\boldsymbol{Q}^{(0,r)}_{\ell+1}$ matrix blocks, for all $r \in \{1, 2, \ldots, \overline{\tau}^{\mathrm{msg}}\}$ and $\ell \geq \overline{\tau}$, have positive entries on their diagonals; we show this using a simple induction argument. Recall that $\boldsymbol{Q}^{(0,r)}_{\ell+1} = \widetilde{\boldsymbol{P}}_0[k+\ell]\boldsymbol{Q}^{(0,r)}_{\ell} + \boldsymbol{Q}^{(1,r)}_{\ell}$. Since $\widetilde{\boldsymbol{P}}_0[k']$ has positive entries on its diagonals at all time-indices k' and $\boldsymbol{Q}^{(1,r)}_{\ell}$ is non-negative, we have that $\boldsymbol{Q}^{(0,r)}_{\ell+1}$ has positive entries on its diagonals if $\boldsymbol{Q}^{(0,r)}_{\ell}$ has positive entries on its diagonals. Now to consider the base cases. We have that $\boldsymbol{Q}_1$ $(\coloneqq \overline{\boldsymbol{P}}[k])$ contains the identity matrix in the $(0,1)$ index block simply by the definition of $\overline{\boldsymbol{P}}[k]$ (4.1). We have that $\boldsymbol{Q}_2$ $(\coloneqq \overline{\boldsymbol{P}}[k+1]\overline{\boldsymbol{P}}[k])$ contains the identity matrix in the $(0,2)$ index block, $\boldsymbol{Q}_3$ $(\coloneqq \overline{\boldsymbol{P}}[k+2]\overline{\boldsymbol{P}}[k+1]\overline{\boldsymbol{P}}[k])$ contains the identity matrix in the $(0,3)$ index block, and so on and so forth for $\boldsymbol{Q}_4$ up to $\boldsymbol{Q}_{\overline{\tau}^{\mathrm{msg}}}$. Therefore, for any integer b between 1 and $\overline{\tau}^{\mathrm{msg}}$, the matrix product $\boldsymbol{Q}_b[k]$ $(\coloneqq \overline{\boldsymbol{P}}[k+b-1]\cdots\overline{\boldsymbol{P}}[k+1]\overline{\boldsymbol{P}}[k])$ contains

the identity matrix in the $(0, b)$ index block (*i.e.*, $\boldsymbol{Q}_b^{(0,b)} = \boldsymbol{I}_{n\times n}$). Since the identity matrix has positive entries on its diagonals, it follows by induction that the matrix blocks $\boldsymbol{Q}_{\ell+1}^{(0,r)}$ for all $r \in \{1, 2, \ldots, \overline{\tau}\}$ and $\ell \geq \overline{\tau}^{\text{msg}} - 1$ have positive entries on their diagonals. Since the result holds for all $\ell \geq \overline{\tau}^{\text{msg}} - 1$, it also holds for all $\ell \geq \overline{\tau}$ (recall $\overline{\tau} \coloneqq \overline{\tau}^{\text{msg}} + \overline{\tau}^{\text{proc}} - 1$).

Therefore, we have shown that for $\ell \geq \overline{\tau}$, the zero/non-zero structure of $\boldsymbol{Q}_{\ell+1}^{(0,0)}$ corresponds to a strongly connected graph, and that the remaining $\boldsymbol{Q}_{\ell+1}^{(0,1)}, \boldsymbol{Q}_{\ell+1}^{(0,2)}, \ldots,$ $\boldsymbol{Q}_{\ell+1}^{(0,\overline{\tau}^{\text{msg}})}$ matrices all have positive entries on their diagonal. This implies that the non-virtual agents form a strongly connected component, and that the virtual agents are all connected to the strongly connected component. Hence, the graph structure contained by $\boldsymbol{Q}_{\ell+1}$ for all $\ell \geq \overline{\tau}$ has only one strongly connected component, and finally we have that the product of $\ell + 1$ consecutive consensus-matrices $\boldsymbol{Q}_{\ell+1}[k]$ $(\coloneqq \overline{\boldsymbol{P}}[k+\ell] \cdots \overline{\boldsymbol{P}}[k+2]\overline{\boldsymbol{P}}[k])$ is (column) stochastic, indecomposable and aperiodic (SIA) for arbitrary k and $\ell \geq \overline{\tau}$.

■

Corollary 1.3 (Weak Convergence of SIA Matrix Products)**.** *As a result of separate interest, we have weak convergence of the consensus matrices* $\overline{\boldsymbol{P}}[k]$.

Proof. The matrix $\overline{\boldsymbol{P}}[k]$ can take no more than $(\overline{\tau}^{\text{msg}} + 2)^{|\mathcal{E}|}$ different matrix values, and therefore can be said to come from a finite collection of matrices, $\mathcal{P}$. The upper bound on the cardinality of the collection $|\mathcal{P}|$ comes from the fact that each agent can transmit on any of the $\overline{\tau}^{\text{msg}}+1$ different delay edges in the augmented graph (0-delay up to $\overline{\tau}^{\text{msg}}$-delay), and has the additional option to not transmit a message on any of its edges (which occurs when the agent is *not* in the *Communication Set*). Therefore,

the number of possible matrix values is upper bounded by the number of possible permutations arising from having $\overline{\tau}^{\text{msg}} + 2$ different transmission options for each edge in the reference graph *i.e.*, $(\overline{\tau}^{\text{msg}} + 2)^{|\mathcal{E}|}$. Using the aforementioned fact and the result of Lemma 1.1, we can proceed to invoke weak convergence from Wolfowitz's Theorem [80] for ergodic matrices. Namely, we have that $\overline{\boldsymbol{P}}[k] \dots \overline{\boldsymbol{P}}[2]\overline{\boldsymbol{P}}[1] \to \boldsymbol{d}[k]\mathbf{1}^\top$ for large k, where $\boldsymbol{d}[k]\mathbf{1}^\top$ is some rank one matrix with all identical columns. Here we have implicitly used the fact that we can invoke Wolfowitz's Theorem for ergodic matrices to the matrix products $\boldsymbol{Q}_{\ell+1}[k]$ for $\ell \geq \overline{\tau}$, rather than the individual $\overline{\boldsymbol{P}}[k]$ (we do this because the matrices $\boldsymbol{Q}_{\ell+1}[k]$ are SIA, but the individual $\overline{\boldsymbol{P}}[k]$ are not necessarily SIA, an important requirement of Theorem [80]). ■

The result of Corollary 1.3 can be used to prove convergence of an unperturbed asynchronous push-sum averaging algorithm by using a similar idea to the proof in [29] for Ratio Consensus with Delays.

Lemma 1.2 (Lower Bound on the Entries of SIA Matrix Products)**.** *It holds for the matrix product* $\boldsymbol{Q}_{\ell+1}[k]$ $(\coloneqq \overline{\boldsymbol{P}}[k+\ell] \cdots \overline{\boldsymbol{P}}[k+1]\overline{\boldsymbol{P}}[k])$ *for all* $\ell \geq n(\overline{\tau}+1)-1$*,* $k \in \mathbb{N}$*, and* $i = 1, 2, \dots, n$*, that*

$$\min_{1 \leq j \leq \overline{\tau}^{msg}+1} [\boldsymbol{Q}_{\ell+1}[k]]_{i,j} \geq \delta_{min} \coloneqq \min_{1 \leq j \leq n} \left(\frac{1}{N_j^{out}}\right)^{n(\overline{\tau}+1)},$$

where n *is the number of non-virtual agents in the network.*

Proof. We can write the matrix product $\boldsymbol{Q}_{n(\overline{\tau}+1)}[k]$ as

$$\begin{aligned}\boldsymbol{Q}_{n(\overline{\tau}+1)}[k] &= \boldsymbol{Q}_{\overline{\tau}+1}[k + (n-1)(\overline{\tau}+1)] \cdots \boldsymbol{Q}_{\overline{\tau}+1}[k + (\overline{\tau}+1)]\boldsymbol{Q}_{\overline{\tau}+1}[k] \\ &= \left(\boldsymbol{Q}_{(n-1)(\overline{\tau}+1)}[k + (\overline{\tau}+1)]\right) \boldsymbol{Q}_{\overline{\tau}+1}[k],\end{aligned}$$

where $\boldsymbol{Q}_{\overline{\tau}+1}[k']$ is defined as the product of $\overline{\tau}+1$ consecutive matrices $\overline{\boldsymbol{P}}[k'+\overline{\tau}]\cdots\overline{\boldsymbol{P}}[k'+1]\overline{\boldsymbol{P}}[k']$. From our intermediate result in Lemma 1.1, we know that, for all $k' \in \mathbb{N}$, the $(0,0)$ index block $\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}[k'] \in \mathbb{R}^{n\times n}$ conforms to a strongly connected reference graph structure. It follows that the product of $n-1$ such matrix blocks $(\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}[k'+(n-2)(\overline{\tau}+1)]\cdots\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}[k'+(\overline{\tau}+1)]\boldsymbol{Q}^{(0,0)}_{\overline{\tau}+1}[k'])$ has positive entries in all positions, and hence $\boldsymbol{Q}^{(0,0)}_{(n-1)(\overline{\tau}+1)}[k+(\overline{\tau}+1)]$ has positive entries in all positions. Furthermore, because each $(0,r)$ index block $\boldsymbol{Q}^{(0,r)}_{\overline{\tau}+1}[k] \in \mathbb{R}^{n\times n}$, for all $r \in \{0,1,\ldots,\overline{\tau}^{\text{msg}}\}$, has positive entries on its diagonals (which we know from our intermediate result in Lemma 1.1), we have that right multiplying $\boldsymbol{Q}_{(n-1)(\overline{\tau}+1)}[k+(\overline{\tau}+1)]$ by the matrix $\boldsymbol{Q}_{\overline{\tau}+1}[k]$ will ensure that each $(0,r)$ index block in the product $\boldsymbol{Q}_{(n-1)(\overline{\tau}+1)}[k+(\overline{\tau}+1)]\boldsymbol{Q}_{\overline{\tau}+1}[k]$ has positive entries in all positions. In other words, each entry in the first n rows of $\boldsymbol{Q}_{n(\overline{\tau}+1)}[k]$ will be positive. Furthermore, since $\boldsymbol{Q}_{n(\overline{\tau}+1)}[k]$ is the product of $n(\overline{\tau}+1)$ consecutive matrices $\overline{\boldsymbol{P}}[k+n(\overline{\tau}+1)-1]\cdots\overline{\boldsymbol{P}}[k+1]\overline{\boldsymbol{P}}[k]$, and the minimum non-zero entry in each $\overline{\boldsymbol{P}}[\cdot]$ is equal to $\min_j(1/N^{\text{out}}_j)$, it follows that the minimum non-zero entry in $\boldsymbol{Q}_{n(\overline{\tau}+1)}[k]$ is greater than or equal to $\min_j(1/N^{\text{out}}_j)^{n(\overline{\tau}+1)}$. If we right multiply the matrix product $\boldsymbol{Q}_{n(\overline{\tau}+1)}[k]$ by any column stochastic matrix $\overline{\boldsymbol{P}}[k-1]$, then the lower bound on the entries in the first n rows of $\boldsymbol{Q}_{n(\overline{\tau}+1)}[k]$ is preserved because the new lower bound lies in the convex hull of each row of $\boldsymbol{Q}_{n(\overline{\tau}+1)}[k]$. Hence, it holds for all $\ell \geq n(\overline{\tau}+1)$ that each entry in the first n rows of $\boldsymbol{Q}_{\ell}[k]$ are bounded below by $(\min_j(1/N^{\text{out}}_j))^{n(\overline{\tau}+1)}$. ■

Lemma 1.3 (Geometric Convergence of Ergodic Matrices)**.** *For arbitrary $k, s \in \mathbb{Z}$ with $k \geq s \geq 0$ we have*

$$\left| \left[\overline{\boldsymbol{P}}[k] \cdots \overline{\boldsymbol{P}}[s+1] \overline{\boldsymbol{P}}[s] \right]_{i,j} - [\phi[k]]_i \right| \leq C q^{k-s}, \quad (i, j = 1, 2, \ldots, n(\overline{\tau}^{msg} + 1))$$

where C is some non-negative constant, $q \in (0, 1)$ and $\phi[k]$ is a stochastic vector.

Proof. The proof here is an application of Wolfowitz's Lemmas [80, Lemmas 1–4] to the asynchronous, time-varying, augmented consensus matrices defined in (4.1). Let $\mathcal{A} \coloneqq \{\boldsymbol{A} \in \mathbb{R}^{n \times n}\}$ be a finite collection of matrices such that the product of any subset of matrices in the collection (possibly with repetition) is stochastic, indecomposable, and aperiodic (SIA). Let $\delta(\boldsymbol{A})$ be a measure of the dissimilarity of the rows of $\boldsymbol{A}$ from one another

$$\delta(A) \coloneqq \max_j \max_{i_1, i_2} \left| [\boldsymbol{A}]_{i_1,j} - [\boldsymbol{A}]_{i_2,j} \right|,$$

and let $\lambda(\boldsymbol{A})$ be the coefficient of ergodicity introduced by Hajnal and Bartlett in [31]

$$\lambda(\boldsymbol{A}) \coloneqq 1 - \min_{i_1, i_2} \sum_j \min \left([\boldsymbol{A}]_{i_1,j}, [\boldsymbol{A}]_{i_2,j} \right).$$

An SIA matrix $\boldsymbol{B}$ is called scrambling if $\lambda(\boldsymbol{B}) < 1$. From [80, Lemma 2] and [31, Lemma 4], it holds that

$$\delta(\boldsymbol{A_1 A_2} \cdots \boldsymbol{A_k}) \leq \Pi_{i=1}^{k} \lambda(\boldsymbol{A}_i). \tag{4.9}$$

Let ψ be the number of possible types (zero/non-zero structures) that an $n \times n$ SIA matrix can take. From [80, Lemmas 3 and 4], all products in the $\boldsymbol{A}$'s of length $\ell \geq (\psi + 1)$ are scrambling. Since any $n \times n$ matrix can take no more than 2^{n^2}

types, ψ is bounded above by 2^{n^2}, however this bound is relatively loose, and can be tightened by taking into account the fact that the matrices under consideration must also conform to the graph structure and be SIA.

Now consider some arbitrary matrices $\boldsymbol{Q}^\top \in \mathbb{R}^{n(\overline{\tau}^{\text{msg}}+1)\times n(\overline{\tau}^{\text{msg}}+1)}$ that come from a finite collection of SIA matrices such that any product in the $\boldsymbol{Q}^\top$'s is also SIA. It follows than any product of length $\psi+1$ in the $\boldsymbol{Q}^\top$'s is scrambling, and since the $\boldsymbol{Q}^\top$'s come from a finite collection of matrices, the number of possible products of length $\psi+1$ is also finite. Hence, there exists some constant $d \in [0,1)$ such that $\delta(\boldsymbol{Q}_i^\top \boldsymbol{Q}_{i+1}^\top \cdots \boldsymbol{Q}_{i+t}^\top) \leq d$ for all i. Here we have implicitly taken d to be the maximum $\lambda(\cdot)$ taken over all products of length $\psi+1$. Applying (4.9) and observing that the floored quotient $\left\lfloor \frac{k-s}{\psi+1} \right\rfloor$ can be regarded as a lower bound on the number of products of length $\psi+1$ in the expression $\boldsymbol{Q}_s^\top \boldsymbol{Q}_{s+1}^\top \ldots \boldsymbol{Q}_k^\top$, we have that

$$\delta(\boldsymbol{Q}_s^\top \boldsymbol{Q}_{s+1}^\top \cdots \boldsymbol{Q}_k^\top) \leq d^{\left\lfloor \frac{k-s}{\psi+1} \right\rfloor} \leq \widetilde{C}\left(d^{\frac{1}{\psi+1}}\right)^{k-s}, \quad (k \geq s \geq 0),$$

where $d \in (0,1)$ and $0 \leq \widetilde{C} < 1/d$ is some non-negative constant. Let $e_j \in \mathbb{R}^{n(\overline{\tau}^{\text{msg}}+1)}$ denote the vector with a 1 in its j^{th} entry, and 0 everywhere else. Substituting the definition for $\delta(\cdot)$ and defining $\tilde{q} \coloneqq d^{\frac{1}{\psi+1}}$ gives

$$\max_{i_1,i_2} \left(\left[\boldsymbol{Q}_s^\top \boldsymbol{Q}_{s+1}^\top \cdots \boldsymbol{Q}_k^\top e_j\right]_{i_1} - \left[\boldsymbol{Q}_s^\top \boldsymbol{Q}_{s+1}^\top \cdots \boldsymbol{Q}_k^\top e_j\right]_{i_2} \right) \leq \widetilde{C}\tilde{q}^{k-s},$$

for all e_j. Defining $\phi[k]$ as a stochastic vector whose j^{th} entry is given by $\phi_j[k] \coloneqq \left[\boldsymbol{Q}_s^\top \boldsymbol{Q}_{s+1}^\top \cdots \boldsymbol{Q}_k^\top e_j\right]_1$ (the j^{th} column of the 1^{st} row of the matrix product), and taking transposes we have

$$\left| [\boldsymbol{Q}_k \cdots \boldsymbol{Q}_{s+1}\boldsymbol{Q}_s]_{j,i} - \phi_j[k] \right| \leq \widetilde{C}\tilde{q}^{k-s},$$

for all $i, j = 1, 2, \ldots, n(\overline{\tau}^{\text{msg}} + 1)$.

To ensure that each $\boldsymbol{Q}$ comes from a finite collection of SIA matrices and all products in the $\boldsymbol{Q}$'s are SIA, we define each $\boldsymbol{Q}_{j'}$ as the product of $\ell \in \{\overline{\tau}+1, \overline{\tau}+2, \ldots, 2\overline{\tau}\}$ consecutive consensus-matrices $\overline{\boldsymbol{P}}[j'+\ell-1]\cdots\overline{\boldsymbol{P}}[j'+1]\overline{\boldsymbol{P}}[j']$ (where the consensus-matrices are as defined in (4.1)). Here we have implicitly used two previous results. The first is Lemma 1.1, which tells us that the product of $\overline{\tau}+1$ or more consecutive matrices, $\overline{\boldsymbol{P}}[k]$, is SIA. The second is Corollary 1.3, which tells us that the matrices $\overline{\boldsymbol{P}}[k]$ come from a finite collection of matrices, and hence the set of products in the $\overline{\boldsymbol{P}}[k]$'s of length $\ell \in \{\overline{\tau}+1, \overline{\tau}+2, \ldots, 2\overline{\tau}\}$ come from a finite collection of matrices. Observing that the floored quotient $\left\lfloor \frac{k'-s'}{\overline{\tau}+1} \right\rfloor$ can be regarded as a lower bound on the number of matrices $\boldsymbol{Q}_{j'}$ in the matrix product $\overline{\boldsymbol{P}}[k']\cdots\overline{\boldsymbol{P}}[s'+1]\overline{\boldsymbol{P}}[s']$, and substituting for the new definition of $\boldsymbol{Q}_{j'}$ gives

$$\left\| \left[\overline{\boldsymbol{P}}[k']\cdots\overline{\boldsymbol{P}}[s'+1]\overline{\boldsymbol{P}}[s']\right]_{j,i} - \phi_j[k'] \right\|_\infty \leq \widetilde{C}\tilde{q}^{\left\lfloor \frac{k'-s'}{\overline{\tau}+1} \right\rfloor} \leq C\left(\tilde{q}^{\frac{1}{\overline{\tau}+1}}\right)^{k'-s'} \leq Cq^{k'-s'},$$

where $C < \widetilde{C}/\tilde{q} < 1/(d\tilde{q}) = 1/(d^{\,(\psi+2)/(\psi+1)})$ is a non-negative constant, and q is defined as $q \coloneqq \tilde{q}^{\frac{1}{\overline{\tau}+1}} = d^{\frac{1}{(\psi+1)(\overline{\tau}+1)}}$, thus $q \in (0,1)$. The result holds for all $k, s \in \mathbb{Z}_+$ such that $k' \geq s' \geq 0$. ■

Lemma 1.4 (Lower Bound on the Stochastic Vector Sequence)**.** *For the vector* $\phi[k] \in \mathbb{R}^{n(\overline{\tau}^{msg}+1)}$ *with* i^{th} *entry given by*

$$\phi_i[k] \coloneqq \left[\overline{\boldsymbol{P}}[k]\cdots\overline{\boldsymbol{P}}[s+1]\overline{\boldsymbol{P}}[s]\right]_{i,1},$$

it holds for all $i = 1, 2, \ldots, n$, *and* $k, s \in \mathbb{Z}$ *such that* $k - s \geq n(\overline{\tau} + 1) - 1$, *that*

$$\phi_i[k] \geq \delta_{min} \coloneqq \min_j \left(\frac{1}{N_j^{out}} \right)^{n(\overline{\tau}+1)},$$

where n *is the number of non-virtual agents in the network.*

Proof. From the definition of ϕ_k we have

$$\phi_i[k] \coloneqq \left[\overline{\boldsymbol{P}}[k] \cdots \overline{\boldsymbol{P}}[s+1] \overline{\boldsymbol{P}}[s] \right]_{i,1},$$

for all $i = 1, 2 \ldots, n(\overline{\tau}+1)$. From Lemma 1.2, we know that $\min_j \left[\overline{\boldsymbol{P}}[k] \ldots \overline{\boldsymbol{P}}[s+1] \overline{\boldsymbol{P}}[s] \right]_{i,j}$ is bounded below by $\delta_{min} \coloneqq \left(\frac{1}{N_j^{\text{out}}} \right)^{n(\overline{\tau}+1)}$ for all $i = 1, 2, \ldots, n$ and $k - s \geq n(\overline{\tau}+1) - 1$. Therefore, it follows that $\phi[k]_i$ is bounded below by δ_{min} for all $i = 1, 2, \ldots, n$ and $k - s \geq n(\overline{\tau} + 1) - 1$. ■

We are now ready to combine all of our previous Lemmas to prove Theorem 1.

Proof of Theorem 1. Let $\overline{\boldsymbol{P}}[k : s]$ denote the matrix product $\overline{\boldsymbol{P}}[k] \cdots \overline{\boldsymbol{P}}[s+1] \overline{\boldsymbol{P}}[s]$; it follows from the definition of the perturbed averaging iteration (4.6) that

$$\boldsymbol{x}[k+1] = \overline{\boldsymbol{P}}[k : 0] \boldsymbol{x}[0] + \sum_{s=1}^{k} \overline{\boldsymbol{P}}[k : s] \boldsymbol{\eta}[s] + \boldsymbol{\eta}[k+1]. \tag{4.10}$$

From column-stochasticity of the consensus-matrices $\overline{\boldsymbol{P}}[k]$, we have that $\mathbf{1}^\top \overline{\boldsymbol{P}}[k] = \mathbf{1}^\top$, and therefore

$$\mathbf{1}^\top \boldsymbol{x}[k+1] = \mathbf{1}^\top \boldsymbol{x}[0] + \sum_{s=1}^{k+1} \mathbf{1}^\top \boldsymbol{\eta}[s]. \tag{4.11}$$

Multiplying each term in the expressions for (4.10) and (4.11) by $\overline{\boldsymbol{P}}[k+1]$ and $\phi[k+1]$ respectively, and taking the difference gives

$$\begin{aligned}\overline{\boldsymbol{P}}[k+1]\boldsymbol{x}[k+1] - \phi[k+1]\mathbf{1}^\top\boldsymbol{x}[k+1] =& (\overline{\boldsymbol{P}}[k+1:0] - \phi[k+1]\mathbf{1}^\top)\boldsymbol{x}[0] \\ &+ \sum_{s=1}^{k+1}(\overline{\boldsymbol{P}}[k+1:s] - \phi[k+1]\mathbf{1}^\top)\boldsymbol{\eta}[s],\end{aligned}$$

for all $k \geq 1$. Defining $\boldsymbol{D}[k:s] \coloneqq \overline{\boldsymbol{P}}[k:s] - \phi[k]\mathbf{1}^\top$ and invoking Lemma 1.3, we have for all $i, j = 1, 2, \ldots, n(\overline{\tau}^{\mathrm{msg}} + 1)$ and $k \geq s \geq 0$, that

$$\left|[\boldsymbol{D}[k:s]]_{ij}\right| \leq Cq^{k-s}, \tag{4.12}$$

where $C \geq 0$ and $q \in (0, 1)$. It follows that

$$\overline{\boldsymbol{P}}[k+1]\boldsymbol{x}[k+1] = \phi[k+1]\mathbf{1}^\top\boldsymbol{x}[k+1] + \boldsymbol{D}[k+1:0]\boldsymbol{x}[0] + \sum_{s=1}^{k}\boldsymbol{D}[k+1:s]\boldsymbol{\eta}[s].$$

From the definitions of $\boldsymbol{w}[k], \boldsymbol{y}[k] \in \mathbb{R}^{n(\overline{\tau}^{\mathrm{msg}}+1)\times d}$ in iterations (4.3) and (4.6) respectively, we have for all $k \geq 1$ that

$$\boldsymbol{w}[k+1] = \overline{\boldsymbol{P}}[k]\boldsymbol{x}[k] \tag{4.13}$$

$$= \phi[k]\mathbf{1}^\top\boldsymbol{x}[k] + \boldsymbol{D}[k:0]\boldsymbol{x}[0] + \sum_{s=1}^{k}\boldsymbol{D}[k:s]\boldsymbol{\eta}[s], \tag{4.14}$$

and

$$\boldsymbol{y}[k+1] = \overline{\boldsymbol{P}}[k:0]\boldsymbol{y}[0] \tag{4.15}$$

$$= \phi[k]\mathbf{1}^\top\boldsymbol{y}[0] + \boldsymbol{D}[k:0]\boldsymbol{y}[0] \tag{4.16}$$

$$= \phi[k]n + \boldsymbol{D}[k:0]\mathbf{1}. \tag{4.17}$$

Substituting these expressions for $\boldsymbol{w}[k+1], \boldsymbol{y}[k+1]$ ((4.14) and (4.17), respectively), into the definition of $\boldsymbol{z}[k+1]$ in (4.5), and subtracting the network-wide average vector $(\mathbf{1}^\top \boldsymbol{x}/n) \in \mathbb{R}^d$ from each $z_i[k+1]$ (the i^{th} row of $\boldsymbol{z}[k+1]$) gives

$$z_i[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} = \frac{\phi_i[k]\mathbf{1}^\top \boldsymbol{x}[k] + [\boldsymbol{D}[k:0]\boldsymbol{x}[0]]_i + \sum_{s=1}^{k}[\boldsymbol{D}[k:s]\boldsymbol{\eta}[s]]_i}{\phi_i[k]n + [\boldsymbol{D}[k:0]\mathbf{1}]_i} - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n},$$

for all $i = 1, 2, \ldots, n(\overline{\tau}^{\text{msg}} + 1)$. Bringing terms to a common denominator and cancelling terms

$$z_i[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} = \frac{n[\boldsymbol{D}[k:0]\boldsymbol{x}[0]]_i + n\sum_{s=1}^{k}[\boldsymbol{D}[k:s]\boldsymbol{\eta}[s]]_i}{n(\phi_i[k]n + [\boldsymbol{D}[k:0]\mathbf{1}]_i)} - \frac{\mathbf{1}^\top \boldsymbol{x}[k][\boldsymbol{D}[k:0]\mathbf{1}]_i}{n(\phi_i[k]n + [\boldsymbol{D}[k:0]\mathbf{1}]_i)}.$$

Since $(\phi_i[k]n + [\boldsymbol{D}[k:0]\mathbf{1}]_i) = [\overline{\boldsymbol{P}}[k:0]\mathbf{1}]_i$, by invoking Lemma 1.2, we have for all $i = 1, 2, \ldots, n$, that

$$n(\phi_i[k]n + [\boldsymbol{D}[k:0]\mathbf{1}]_i) = n[\overline{\boldsymbol{P}}[k:0]\mathbf{1}]_i \geq n^2\delta_{min},$$

where $\delta_{min} > 0$ is a finite positive constant. Thus, for all $i = 1, 2, \ldots, n$ and $k \geq 1$,

$$\begin{aligned}\left\| z_i[k+1] - \tfrac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} \right\|_1 \leq \; & \tfrac{1}{n\delta_{min}} \left(\max_j |[\boldsymbol{D}[k:0]]_{ij}|\right) \|\boldsymbol{x}[0]\|_1 \, n \\ & + \tfrac{1}{n\delta_{min}} \textstyle\sum_{s=1}^{k} \left(\max_j |[\boldsymbol{D}[k:s]]_{ij}|\right) \|\boldsymbol{\eta}[s]\|_1 \, n \\ & + \tfrac{1}{n^2\delta_{min}} \left\|\mathbf{1}^\top \boldsymbol{x}[k]\right\|_1 \left(\max_j |[\boldsymbol{D}[k:0]]_{ij}|\right) n^2,\end{aligned}$$

where $\|\boldsymbol{\eta}[s]\|_1$ and $\|\boldsymbol{x}[0]\|_1$ denote 1-matrix norms (maximum absolute column sums). Using the entry-wise decay bound on $|[\boldsymbol{D}[k:s]]_{ij}|$ in (4.12) gives us

$$\left\| z_i[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} \right\|_1 \leq \frac{C}{\delta_{min}} q^k \|\boldsymbol{x}[0]\|_1 + \frac{C}{\delta_{min}} \left(\sum_{s=1}^{k} q^{k-s} \|\boldsymbol{\eta}[s]\|_1 + \left\|\mathbf{1}^\top \boldsymbol{x}[k]\right\|_1 q^k \right),$$

for all $i = 1, 2, \ldots, n$ and all $k \geq 1$. Also, by apply the norm operator to each side we have

$$\left\|\mathbf{1}^\top \boldsymbol{x}[k]\right\|_1 \leq \|\boldsymbol{x}[0]\|_1 + \sum_{s=1}^{k} \|\boldsymbol{\eta}[s]\|_1 .$$

Because $q \in (0,1)$, it follows that $\sum_{s=1}^{k} q^k \|\boldsymbol{\eta}[s]\|_1 \leq \sum_{s=1}^{k} q^{k-s} \|\boldsymbol{\eta}[s]\|_1$, and therefore

$$\left\| z_i[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} \right\|_1 \leq \frac{2C}{\delta_{min}} \left(q^k \|\boldsymbol{x}[0]\|_1 + \sum_{s=1}^{k} q^{k-s} \|\boldsymbol{\eta}[s]\|_1 \right),$$

for all $i = 1, 2, \ldots, n$ and $k \geq 1$. From our block matrix definition of $\boldsymbol{z}[k] \in \mathbb{R}^{n(\overline{\tau}^{\text{msg}}+1)\times d}$ as

$$\boldsymbol{z}[k] \coloneqq \begin{bmatrix} \boldsymbol{z}^{(0)}[k] \\ \boldsymbol{z}^{(1)}[k] \\ \vdots \\ \boldsymbol{z}^{(\overline{\tau})}[k]) \end{bmatrix},$$

where each $\boldsymbol{z}^{(r)}[k] \in \mathbb{R}^{n\times d}$ is a block matrix storing a copy of the variable z at all the delay-r agents at time k, we have that the first n rows of $\boldsymbol{z}$ correspond to the first n rows of the $\boldsymbol{z}^{(0)}$ matrix block. Therefore, for $\widetilde{C} \coloneqq 2C/\delta_{min}$, it holds that

$$\left\| z_i^{(0)}[k+1] - \frac{\mathbf{1}^\top \boldsymbol{x}[k]}{n} \right\|_1 \leq \widetilde{C} q^k \|\boldsymbol{x}[0]\|_1 + \widetilde{C} \sum_{s=1}^{k} q^{k-s} \|\boldsymbol{\eta}[s]\|_1 ,$$

for all $i = 1, 2, \ldots, n$ and $k \geq 1$. ■

CHAPTER 5
Asynchronous Subgradient Push

In this chapter we present and prove convergence of an asynchronous version of the synchronous Subgradient-Push optimization algorithm. Our definition of asynchrony in the multi-agent optimization setting implies that agents do not wait for each other to complete computations, nor do they wait for messages to be sent/received before moving on to the next step in the algorithm. Given that agents do not necessarily perform updates at the same times or at the same rate and that messages between agents may be subject to delays, it follows that agents may perform a drastically different number of gradient steps over any time interval and, in particular, may perform updates using outdated messages.

5.1 Formulation of Asynchronous Subgradient Push

We now make explicit our asynchronous iterations. At all time-indices k, each agent, $v_i^{(r)}$, locally maintains the variables $w_i^{(r)}[k], z_i^{(r)}[k], x_i^{(r)}[k] \in \mathbb{R}^d$, and $y_i^{(r)}[k] \in \mathbb{R}_+$. To analyze the Asynchronous Gradient Push Optimization algorithm from a global perspective, we us the matrix-based formulation provided in Algorithm 7. The matrix-based formulation describes how the agents' values evolve at some time $t[k+1] \in T = \{t[1], t[2], t[3], \dots, \}$, a time at which one or more agents perform an update. The asynchronous communication dynamics are accounted for in the consensus-matrices $\overline{\boldsymbol{P}}[\cdot]$, and the matrix-valued function $\nabla\overline{\boldsymbol{F}}[k+1] \in \mathbb{R}^{n(\overline{\tau}^{\text{msg}}+1)\times d}$ is

Algorithm 7 Asynchronous Gradient Push Optimization

$$\boldsymbol{w}[k+1] = \overline{\boldsymbol{P}}[k]\boldsymbol{x}[k] \tag{5.1}$$
$$y[k+1] = \overline{\boldsymbol{P}}[k]y[k] \tag{5.2}$$
$$\boldsymbol{z}[k+1] = \mathrm{diag}(y[k+1])^{-1}\boldsymbol{w}[k+1] \tag{5.3}$$
$$\boldsymbol{x}[k+1] = \boldsymbol{w}[k+1] - \alpha[k+1]\nabla\overline{\boldsymbol{F}}[k+1] \tag{5.4}$$

defined as

$$\nabla\overline{\boldsymbol{F}}[k+1] \coloneqq \begin{bmatrix} \nabla\boldsymbol{f}^{(0)}(\boldsymbol{z}^{(0)}[k+1]) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}.$$

In particular, the notation $\nabla\boldsymbol{f}^{(0)}(\boldsymbol{z}^{(0)}[k+1]) \in \mathbb{R}^{n\times d}$ denotes a block matrix with i^{th} row equal to

$$\delta_i[k+1]\nabla f_i^{(0)}(z_i^{(0)}[k+1]).$$

The scalar-valued function $\delta_i[\cdot]$ is a $0,1$–indicator that is equal to 1 at some time before agent v_i completes an update, and is equal to 0 otherwise. The *non-virtual* agent initializations are $x_i^{(0)}[0] \in \mathbb{R}^d$, and $y_i^{(0)}[0] = 1$. The *virtual* agent initializations are $x_i^{(r)}[0] = \mathbf{0}$, and $y_i^{(r)}[0] = 0$ (for all $r \neq 0$).[1]

[1] Note, given the initializations, the virtual agents could potentially have $z_i^{(r)}[k+1]/0$ (division by zero) in update equation (5.3), but this is a non-issue since $z_i^{(r)}$ (for all $r \neq 0$) is never used to produce gradients according to the definition of the gradient matrix $\nabla\overline{\boldsymbol{F}}[k+1]$, and anyways, the virtual-agents are only introduced for analysis purposes.

We use a similar notation to synchronous Subgradient Push [57]: w is the push-sum numerator, and y is the push-sum weight. Nonetheless there are a few noteworthy differences between iterations (5.1)–(5.4) and the synchronous subgradient iterations. First, the consensus matrices in the asynchronous update are defined for an augmented state, and in particular, model the *asynchronous communication* between agents. Second, the gradient step (5.4) in the asynchronous iteration contains the presence of $0, 1$–indicator functions. If agent v_i is in the communication set at some times $t[k']$ and $t[s']$, and at no other time in between, where $t[k'] > t[s'] \geq 0$, then the length of the interval $(t[s'], t[k'])$ can be thought of as a processing delay experienced by agent v_i at time $t[k']$, during which, agent v_i neither sends nor processes any new messages — the only messages that are processed in this time-interval are those that were read from the receive-buffer when the agent was last in the communication set (at time $t[s']$). By the end of the time interval, agent v_i completes a **Local Computation**, and hence a gradient step. Mathematically, this is equivalent to setting the gradient activator, $\delta_i[k]$, to 1 once — and only once — at some time in the interval $(t[s'], t[k'])$. Since agent v_i neither sends nor processes any new messages in the interval $(t[s'], t[k'])$, the specific time in the interval at which $\delta_i[\cdot]$ is set to 1 does not matter. At all other times in the interval, $\delta_i[\cdot]$ will be set to 0. For example, if agent v_i completes its first update at time $t[2]$, then $\delta_i[\cdot]$ can be set to 1 at time $t[0]$, and set to 0 at times $t[1]$ and $t[2]$. Alternatively, $\delta_i[\cdot]$ can be set to 1 at time $t[1]$, and set to 0 at times $t[0]$ and $t[2]$. Another option is to set $\delta_i[\cdot]$ to 1 at time $t[2]$, and set to 0 at times $t[0]$ and $t[1]$. From an analysis perspective, all permutations produce the same result.

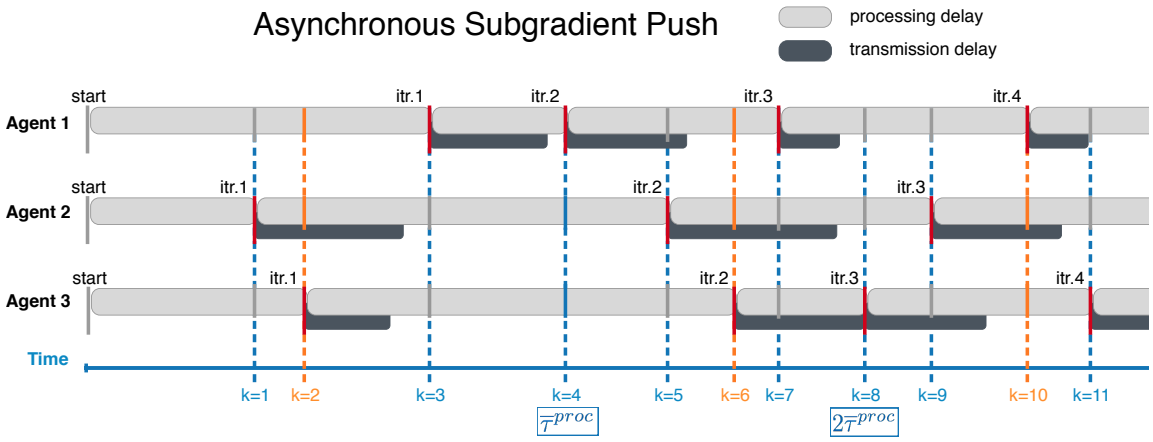


Figure 5–1: (Best viewed in colour). Example of agent updates in an Asynchronous Gradient Push procedure with a maximum time-index processing delay $\overline{\tau}^{\text{proc}} = 4$. The time-index, k, increments by 1 each time an agent performs an update (completes a **Local Computation**). At the end of each update, the updating agent initiates a message transmission to its neighbours and proceeds with its local computation. The diagram depicts the local iteration increments (in red), the time axis with delineated $\overline{\tau}^{\text{proc}}$ time-index increments (in blue), and one possible choice for a subsequence of partially overlapping computations (in orange). Note that each time-index in the subsequence of partially overlapping computations could potentially correspond to a different local iteration at each agent. For example, $k = 2$ corresponds to Agent 1's first iteration, Agent 2's second iteration, and Agent 3's second iteration.

Remark 4 (Subsequence of Partially Overlapping Computations)**.** *In our subsequent analysis we show that a local subsequence of the iterates at each agent converges to a neighbourhood of the global minimum. To construct such a subsequence, we define a* subsequence of partially overlapping computations*: By the bounded processing delay assumption it follows that each agent, v_j, needs to set its indicator, $\delta_j[\cdot]$, to* 1 *at least once every $\overline{\tau}^{proc}$ time indices. Hence, in every $\overline{\tau}^{proc}$ time-index interval there is at least one time-index during which all agents can simultaneously set their gradient activator to* 1*. We refer to these time-indices as the elements of a* subsequence of partially overlapping computations*, which we denote by $\{b_k\}$. Since such a time index must*

occur at least once in every $\overline{\tau}^{proc}$ interval, it follows that the time-index difference between successive subsequence terms is at most $\overline{\tau}^{proc}$ (i.e., $b_{k+1} - b_k \leq \overline{\tau}^{proc}$ for all $k \geq 0$). Figure 5–1 shows an example of such a subsequence of partially overlapping computations.

5.2 Main Results

Assumption 2 (Existence, Convexity, and Smoothness)**.** *A minimizer of* (1.1) *exists, and the local objective functions are strongly-convex and have Lipschitz-continuous gradients. Precisely:*

1. $\operatorname{argmin}_x F(x) \neq \emptyset$.
2. *Each function $f_i(x) : \mathbb{R}^d \mapsto \mathbb{R}$ is m_i-strongly convex, and has M_i-Lipschitz continuous gradients.*

Let $m \coloneqq \min(m_i)$, and $M \coloneqq \max(M_i)$, where m_i and M_i are as defined in Assumption 2. In addition, let $N_{max}^{\text{out}} \coloneqq \max_{1 \leq j \leq n} N_j^{\text{out}}$ represent the maximum number of out-neighbours associated to any non-virtual agent. Let $\overline{x}[k] \coloneqq \mathbf{1}^\top \boldsymbol{x}[k]/n$ be the mutual time-wise average of the variable x at time-index k, and let $x^\star \coloneqq \operatorname{argmin} \overline{\boldsymbol{F}}(x)$, the global minimizer.

Theorem 2 (Convergence of Asynchronous Gradient Push for Diminishing Step–Size)**.** *If Assumption 1 and Assumption 2 are satisfied, and the (strictly positive) step-size sequence $\{\alpha[k]\}$ is non-increasing and satisfies*

$$\sum_{k=1}^{\infty} \alpha[k] = \infty, \quad \sum_{k=1}^{\infty} \alpha^2[k] < \infty, \quad \alpha[0] \leq \left(\frac{m}{2M^2}\right)\left(\frac{1}{N_{max}^{out}}\right)^{n(\overline{\tau}+1)},$$

then for all $i = 1, 2, \ldots, n$,

$$\liminf_{k\to\infty} \|z_i[k] - x^\star\| < \frac{L}{m}\sqrt{2(\overline{\tau}^{proc} - 1)(n-1)},$$

Remark. *The result of Theorem 2 states that by using a diminishing step-size sequence, the agents running the Asynchronous Gradient Push optimization algorithm are guaranteed to agree on a solution that converges to a neighbourhood of the global minimizer, where the size of the neighbourhood depends on the maximum time-index processing delay, the modulus of strong-convexity, and the Lipschitz constant. Hence, as* $\overline{\tau}^{proc}$ *decays to* 1 *(i.e., the algorithm operates* semi-synchronously*; agents wait for each other to complete updates, but don't wait for messages to be sent/received), the* z_i *terms at each agent converge to the global minimizer, even if the communication delays do not go zero.*

Remark. *It is worth pointing out that the upper bound on the step-size is quite small in practice and perhaps overly conservative. In our numerical simulations we observe that there exist much larger step-sizes that still lead to convergence. In fact, one can show that the result of Theorem 2 still holds if, rather than having* $\alpha[0] \leq \left(\frac{m}{2M^2}\right)\left(\frac{1}{N_{max}^{out}}\right)^{n(\overline{\tau}+1)}$*, it (instead) holds that* $\alpha[k] \leq \left(\frac{m}{2M^2}\right) y_i[k]$ *for all* k *sufficiently large, where* $y_i[k]$ *is the push-sum weight held locally by agent* v_i *at time* k*. If the agents work at roughly the same rate then* $y_i[k] \approx 1$*. This remark applies wherever the inequality* $\alpha[0] \leq \left(\frac{m}{2M^2}\right)\left(\frac{1}{N_{max}^{out}}\right)^{n(\overline{\tau}+1)}$ *appears in any theorem or lemma statements throughout the rest of this thesis.*

Theorem 3 (Convergence of Asynchronous Gradient Push for Constant Step-Size)**.** *If Assumption 1 and Assumption 2 are satisfied, and the step-size,* $\alpha > 0$*, is a*

constant satisfying

$$\alpha \leq \min\left\{\left(\frac{m}{2M^2}\right)\left(\frac{1}{N_{max}^{out}}\right)^{n(\overline{\tau}+1)}, \frac{((n-1)(\overline{\tau}^{proc}-1))^3}{2n(m(\overline{\tau}^{proc})^2+\frac{2\overline{\tau}^{proc}M\sqrt{d}C}{(1-q)})}\right\},$$

then for all $i = 1, 2, \ldots, n$

$$\liminf_{k\to\infty} \|z_i[k] - x^\star\| < \frac{L}{m}\sqrt{2(\overline{\tau}^{proc}-1)(n-1)} + \alpha\frac{\widetilde{C}L}{1-q},$$

where $\widetilde{C}$ *is the constant defined in Corollary 1.2.*

Theorem 4 (Convergence of Semi-Synchronous Gradient Push for Constant Step–Size)**.** *Suppose* $\overline{\tau}^{proc} = 1$ *or* $n = 1$ *and Assumption 1 and Assumption 2 are satisfied. For any* $\rho > 0$*, if the step-size,* $\alpha > 0$*, is constant and satisfies*

$$\alpha \leq \min\left\{\left(\frac{m}{2M^2}\right)\left(\frac{1}{N_{max}^{out}}\right)^{n(\overline{\tau}+1)}, \frac{m\rho^2}{nL^2\left((\overline{\tau}^{proc})^2+\frac{2\overline{\tau}^{proc}MC\sqrt{d}}{m(1-q)}\right)}\right\},$$

then for all $i = 1, 2, \ldots, n$*,*

$$\liminf_{k\to\infty} \|z_i[k] - x^\star\| < \sqrt{\frac{2L\rho}{m}} + \alpha\frac{CL}{1-q}.$$

Remark 5. *The* $\mathcal{O}\left(\sqrt{\rho}\right)$ *term is the neighbourhood to which the mutual time-wise average converges, and the* $\frac{\alpha CL}{1-q}$ *term is due to the worst-case consensus disagreement between agents. The step-size bound depends on the desired size of the neighbourhood of convergence, and the connectivity and asynchrony in the multi-agent network. Interestingly so, the communication delays do not play a role in this asymptotic convergence result. However one would expect communication delays to play an important role in any derived convergence rate, as is the case in Theorem 1, where the delays*

directly affect the geometric rate at which the asynchronous Perturbed Push-Sum algorithm converges to the mutual time-wise average.

5.3 Analysis

In order to prove convergence of Algorithm 7 for both the constant and diminishing step-size cases, we first observe that Algorithm 7 (Asynchronous Subgradient Push Optimization) can be reduced to Algorithm 6 (Asynchronous Perturbed Push-Sum Averaging) by letting the perturbation term, $\boldsymbol{\eta}[k]$, equal the gradient term, $-\alpha[k+1]\nabla\overline{\boldsymbol{F}}[k+1]$, defined in iteration (5.4). That is, by regarding the gradient term as a single exogenous perturbation to an asynchronous push-sum averaging procedure, and showing that these perturbations remain bounded, we have that the iterate sequences at all the non-virtual agents (the $z_i^{(0)}[k]$ terms) converge to a neighbourhood of the mutual time-wise average in the constant step-size case, and to the exact average in the diminishing step-size case. Then by showing that the mutual time-wise average converges to a point in a neighbourhood of the minimizer, we have the desired result. The main challenge in the analysis is to show that the perturbation terms (the gradients) do indeed remain bounded, and that the mutual time-wise average converges to a point in a neighbourhood of the minimizer.

5.3.1 Preliminaries

Theorem 5 (Bounded Iterates and Gradients)**.** *If Assumption 2 is satisfied and, for all agents v_i, the terms in the step-size sequence $\{\alpha_i[k]\}$ satisfy*

$$\alpha_i[k] \leq \frac{m}{2M^2} y_i[k] \qquad (\textit{for all } k \in \mathbb{N}),$$

then there exist $D, L \in \mathbb{R}_{++}$ *such that,*

$$\sup_k \|\nabla f_i(z_i[k])\| \leq L,$$

$$\sup_k \|\overline{x}[k]\| \leq D.$$

Remark. *One can also show that the result of Theorem 5 still holds if, rather than requiring all step-sizes in the step-size sequence* $\{\alpha_i[k]\}$ *to satisfy the inequality* $\alpha_i[k] \leq \frac{m}{2M^2} y_i[k]$*, there exists some finite* $k_0 \in \mathbb{N}$*, such that the inequality is satisfied for all* $k \geq k_0$*.*

Before proving Theorem 5 we first establish a few key lemmas.

Lemma 5.1. *Let* $f : \mathbb{R}^d \to \mathbb{R}$ *be an* m*-strongly convex function with* M*-Lipschitz-continuous gradients. Also, let* $u, v \in \mathbb{R}^d$ *be related as*

$$u = v - \alpha \nabla f(v)$$

for some $\alpha \in [0, m/(2M^2)]$*. Then there exists a compact set* $X \subset \mathbb{R}^d$ *such that*

$$\|u\| \leq \begin{cases} \|v\|, & (v \notin X) \\ R, & (v \in X) \end{cases}$$

where

$$R = \max_{z \in X} \{\|z\| + \alpha \|\nabla f(z)\|\}.$$

The proof of Lemma 5.1 is very similar to that in [55, Lemma 3], where the analysis is conducted in the setting of stochastic gradients; we adapt the analysis here to provide a looser constraint on the step-size by making-do without the additive stochastic noise assumption in the update equation.

Proof. Strong convexity of f implies

$$\langle \nabla f(v), v \rangle \geq f(v) - f(0) + \frac{m}{2} \|v\|^2 . \tag{5.5}$$

Using the definition of u and substituting in the inequality (5.5), we have

$$\begin{aligned} \|u\|^2 &= \|v\|^2 - 2\alpha \langle \nabla f(v), v \rangle + \alpha^2 \|\nabla f(v)\|^2 \\ &\leq (1 - \alpha m) \|v\|^2 - 2\alpha (f(v) - f(0)) + \alpha^2 \|\nabla f(v)\|^2 . \end{aligned} \tag{5.6}$$

Using Lipschitz-continuity of the gradient, and Young's Inequality, $(a+b)^2 \leq 2a^2 + 2b^2$,

$$\|\nabla f(v)\|^2 \leq (\|\nabla f(v) - \nabla f(0)\| + \|\nabla f(0)\|)^2 \leq 2M^2 \|v\|^2 + 2 \|\nabla f(0)\|^2 . \tag{5.7}$$

Substituting (5.7) back into the expression in (5.6)

$$\begin{aligned} \|u\|^2 &\leq (1 - \alpha m) \|v\|^2 - 2\alpha (f(v) - f(0)) + 2\alpha^2 M^2 \|v\|^2 + 2\alpha^2 \|\nabla f(0)\|^2 \\ &= (1 - \alpha (m - 2\alpha M^2)) \|v\|^2 - 2\alpha (f(v) - f(0)) + 2\alpha^2 \|\nabla f(0)\|^2 . \end{aligned} \tag{5.8}$$

For all $\alpha \in [0, m/(2M^2)]$ the expression (5.8) simplifies to

$$\begin{aligned} \|u\|^2 &\leq \|v\|^2 - 2\alpha (f(v) - f(0)) + 2\alpha^2 \|\nabla f(0)\|^2 \\ &= \|v\|^2 - 2\alpha \left(f(v) - f(0) - \alpha \|\nabla f(0)\|^2 \right) . \end{aligned}$$

Define the level set of f,

$$X \coloneqq \left\{ z \;\middle|\; f(z) \leq f(0) + \frac{m}{2M^2} \|\nabla f(0)\|^2 \right\} .$$

Since f is finite-valued and strongly-convex (hence convex), it is continuous on the interior of its domain, and hence lower-semicontinuous, which gives us closed level-sets

(cf. [35]). Also, since f is strongly-convex, it is also supercoercive (hence coercive), and thus has bounded level-sets (cf. [35]). Putting these two pieces together, it follows that the level-sets of a finite-valued strongly-convex function are compact, thus we have that the set X is compact. If $v \notin X$, we can get rid of the second term in the inequality, leaving us with

$$\|u\|^2 \leq \|v\|^2 .$$

If $v \in X$, then from the definition of u we have

$$\|u\|^2 \leq \|v\|^2 + \alpha \|f(v)\|^2 .$$

■

Now for convenience, define $\mathcal{I}[k]$ to be the set of indices corresponding to agents with non-zero push-sum weights at time k:

$$\mathcal{I}[k] \coloneqq \{i \in \mathbb{N} \mid 1 \leq i \leq n(\overline{\tau}^{\text{msg}} + 1), y_i[k] \neq 0\} .$$

Lemma 5.2. *For all $k \geq 1$ and $i \in \mathcal{I}[k+1]$, the push-sum update* (5.3) *can instead be written as*

$$z_i[k+1] = \sum_{x_j[k] \neq 0, j=1}^{n(\overline{\tau}^{msg}+1)} Q_{ij}[k] \left(\frac{x_j[k]}{y_j[k]} \right),$$

where the i^{th} row of the matrix $\boldsymbol{Q}[k] \in \mathbb{R}^{n(\overline{\tau}^{msg}+1) \times n(\overline{\tau}^{msg}+1)}$ is stochastic.

Proof. From the first three update equations in Algorithm 7; we have that for all $k \geq 1$ and $i \in \mathcal{I}[k+1]$,

$$w_i[k+1] = \sum_{j=1}^{n(\overline{\tau}^{\mathrm{msg}}+1)} P_{ij}[k]x_j[k]$$
$$y_i[k+1] = \sum_{j=1}^{n(\overline{\tau}^{\mathrm{msg}}+1)} P_{ij}[k]y_j[k]$$
$$z_i[k+1] = \frac{w_i[k+1]}{y_i[k+1]}.$$

Notice that without any loss of generality, we can rewrite the w_i update by excluding the terms in the sum where $x_j[k] = 0$ (since the entries of $\boldsymbol{P}[k]$ are always finite)

$$w_i[k+1] = \sum_{x_j[k]\neq 0, j=1}^{n(\overline{\tau}^{\mathrm{msg}}+1)} P_{ij}[k]x_j[k].$$

Since $w_i[k+1] = y_i[k+1]z_i[k+1]$, we can further rewrite the w_i update as

$$y_i[k+1]z_i[k+1] = \sum_{x_j[k]\neq 0, j=1}^{n(\overline{\tau}^{\mathrm{msg}}+1)} P_{ij}[k]y_j[k]x_j[k]/y_j[k],$$

where we have multiplied the right hand side by $1 = y_j[k]/y_j[k]$. Note that for $j > n$ (the virtual agent indices), it is possible that $y_j[k] = 0$, however this only happens at initialization, or if the virtual agent has forwarded all of its information to the next agent in the delay daisy-chain, and has not received any new information since (cf. Lemma 1.4). Therefore, the event $y_j[k] = 0$ *always* coincides with $x_j[k] = 0$ (though the converse is not necessarily true), and since these terms are excluded anyways,

the summation is well-defined. Hence for all $i \in \mathcal{I}[k+1]$, we have

$$z_i[k+1] = \sum_{x_j[k]\neq 0, j=1}^{n(\overline{\tau}+1)} (1/y_i[k+1]) P_{ij}[k] y_j[k] x_j[k]/y_j[k],$$

Defining the matrix $\boldsymbol{Q}[k] \in \mathbb{R}^{n(\overline{\tau}^{\text{msg}}+1)\times n(\overline{\tau}^{\text{msg}}+1)}$ as

$$Q_{ij}[k] \coloneqq \begin{cases} \frac{1}{y_i[k+1]} P_{ij}[k] y_j[k], & y_i[k+1] \neq 0 \\ 0, & \text{otherwise,} \end{cases}$$

gives for all $i \in \mathcal{I}[k+1]$

$$z_i[k+1] = \sum_{x_j[k]\neq 0, j=1}^{n(\overline{\tau}^{\text{msg}}+1)} Q_{ij}[k] x_j[k]/y_j[k].$$

To see that for all $i \in \mathcal{I}[k+1]$ the i^{th} row of $\boldsymbol{Q}[k]$ is stochastic, notice that

$$\sum_{j=1}^{n(\overline{\tau}^{\text{msg}}+1)} Q_{ij}[k] = \frac{1}{y_i[k+1]} \sum_{j=1}^{n(\overline{\tau}^{\text{msg}}+1)} P_{ij}[k] y_j[k] = \frac{1}{y_i[k+1]} y_i[k+1] = 1.$$

■

Proof of Theorem 5. From the result of Lemma 1.4, we have that each entry in $\boldsymbol{y}^{(0)}[k]$ is bounded below by $\delta_{min} > 0$ at all times $k \geq 1$. That is, for all $i = 1, 2, \ldots, n$ and $k \geq 1$, it holds that

$$y_i[k] \geq \delta_{min} \coloneqq \left(\frac{1}{N_{max}^{\text{out}}}\right)^{n(\overline{\tau}+1)} > 0.$$

Hence, we can rewrite the index set $\mathcal{I}[k]$ as

$$\mathcal{I}[k] = \{1, \ldots, n\} \cup \left\{ \hat{i} \in \mathbb{N} \;\middle|\; n < \hat{i} \leq n(\overline{\tau}^{\text{msg}} + 1), y_i[k] \neq 0 \right\}.$$

Now from the definition of the variable x in (5.4) and the variable z in (5.3), we have that for all $i = 1, 2, \ldots, n$ and $k \geq 1$,

$$\begin{aligned} x_i[k+1] &= w_i[k+1] - \alpha[k+1]\delta_i[k+1]\nabla f_i[k+1] \\ &= y_i[k+1]\left(z_i[k+1] - \frac{\alpha[k+1]\delta_i[k+1]}{y_i[k+1]}\nabla f_i[k+1]\right), \end{aligned}$$

which can be rewritten as

$$\frac{x_i[k+1]}{y_i[k+1]} = z_i[k+1] - \frac{\alpha[k+1]\delta_i[k+1]}{y_i[k+1]}\nabla f_i[k+1].$$

Recall that $\nabla f_i[k+1]$ is the gradient of f_i evaluated at $z_i[k+1]$, and $\delta_i[k+1]$ is a $0,1$-indicator. By assumption, $\alpha[k+1] \in (0, y_i[k+1]m/(2M^2)]$; therefore we can apply the results of Lemma 5.1, with u defined as

$$u_i[k] \coloneqq \frac{x_i[k]}{y_i[k]}.$$

Therefore, for all $i = 1, 2, \ldots, n$ and $k \geq 1$,

$$\left\| \frac{x_i[k+1]}{y_i[k+1]} \right\| \leq \begin{cases} \|z_i[k+1]\|, & (z_i[k+1] \notin X) \\ R_i, & (z_i[k+1] \in X). \end{cases} \tag{5.9}$$

As for the virtual agents (indices $j = n+1, n+2, \ldots, n(\overline{\tau}^{\mathrm{msg}} + 1)$), it holds for all $k \geq 1$ that when $z_j[k+1]$ is well defined (*i.e.*, $y_j[k+1] \neq 0$), the following holds

$$\left\| \frac{x_j[k+1]}{y_j[k+1]} \right\| = \|z_j[k+1]\|. \tag{5.10}$$

Expression (5.10) follows from the definition of the $z_j[k]$ update in (5.3) and the fact that $f_j(\cdot) = 0$ for all $j \in \{n+1, \ldots, n(\overline{\tau}^{\mathrm{msg}} + 1)\}$.

From Lemma 5.2, we know that for all $i \in \mathcal{I}[k+1]$, the variable $z_i[k+1]$ is in the convex hull of $\left\{ \frac{x_j[k]}{y_j[k]} \;\middle|\; x_j[k] \neq 0 \right\} \subseteq \left\{ \frac{x_j[k]}{y_j[k]} \;\middle|\; y_j[k] \neq 0 \right\}$; invoking this property and using the convexity of the norm and the bounds in (5.9) and (5.10), we have that for all $k \geq 1$, and for all $i \in \mathcal{I}[k+1]$,

$$\begin{aligned} \|z_i[k+1]\| \leq \max_{\{j \mid y_j[k] \neq 0\}} \left\| \frac{x_j[k]}{y_j[k]} \right\| \leq \max\{ &\max_{1 \leq j \leq n} \|z_j[k]\|, \max_{1 \leq j \leq n} R_j, \\ &\max_{\{n < j \leq n(\overline{\tau}^{\mathrm{msg}}+1) \mid y_j[k] \neq 0\}} \|z_j[k]\| \}. \end{aligned} \tag{5.11}$$

Recall that

$$\mathcal{I}[k] = \{1, \ldots, n\} \cup \left\{ \hat{i} \in \mathbb{N} \;\middle|\; n < \hat{i} \leq n(\overline{\tau}^{\mathrm{msg}} + 1), y_i[k+1] \neq 0 \right\},$$

and therefore, the bound in (5.11) can be applied recursively to the each of the $\|z_j[k]\|$ terms on the right hand side of inequality (5.11). Since $\|z_i[1]\|$ is deterministic and bounded for all $i \in \{1, \ldots, n\}$, and since $y_j[1] = 0$ for all $j \in \{n, \ldots n(\overline{\tau}^{\mathrm{msg}} + 1)\}$ — hence $\{\|z_j[1]\| \mid n < j \leq n(\overline{\tau}^{\mathrm{msg}} + 1), y_j[1] \neq 0\}$ is the empty set — it follows through induction that $\|z_i[k]\|$ is bounded for all $i \in \{1, \ldots, n(\overline{\tau}^{\mathrm{msg}} + 1)\}$ and $k \geq 1$. Specifically

$$\|z_i[k]\| \leq \max\{ \max_{1 \leq i \leq n} \|z_i[1]\|, \max_{1 \leq i \leq n} R_i \} =: \widetilde{D}.$$

To see that the gradients are also bounded, just invoke their Lipschitz-continuity: Let $z_i^\star$ be the minimizer of f_i, and let $\nabla f_i^\star$ be the gradient of f_i evaluated at $z_i^\star$. Then for all $i \in \{1, \ldots, n\}$ and $k \geq 1$

$$\|\nabla f_i[k+1]\| = \|\nabla f_i[k+1] - \nabla f_i^\star\| \leq M_i \|z_i[k+1] - z_i^\star\| \leq M(\widetilde{D} + \|z_i^\star\|),$$

Therefore, there exists an $L \in \mathbb{R}_{++}$ such that for all $i \in \{1, \ldots, n\}$ and $k \geq 1$,

$$\|\nabla f_i[k+1]\| \leq L \leq \max_{1 \leq i \leq n} M_i(\widetilde{D} + \|z_i^\star\|).$$

The result also holds trivially for all $i > n$ (the virtual agent indices) since the corresponding gradients equal 0 by definition. It follows that $\|\overline{x}[k]\|$ is also bounded for all $k \geq 1$. In particular, using the result from Theorem 1 and substituting in the bounds for $\|z_i[k+1]\|$ and $\|\nabla f_i[k+1]\|$ we have

$$\|\overline{x}[k]\| \leq \widetilde{D} + \max_{1 \leq i \leq n} \left(C \|x_i[1]\| + C\alpha_i[1] \frac{L}{1-q} \right) =: D,$$

where $\alpha_i[1]$ is the initialized step-size at agent v_i. Thus we have that the iterates and gradients at each agent remain bounded, the desired result. ■

The following lemmas, together with Theorem 5, will be used in the convergence proofs of Asynchronous Subgradient Push.

Lemma 5.3. *If Assumption 2 is satisfied, and the step-size sequence $\{\alpha[k]\}$ is non-increasing and satisfies*

$$\alpha[0] \leq \left(\frac{m}{2M^2} \right) \left(\frac{1}{N_{max}^{out}} \right)^{n(\overline{\tau}+1)},$$

and defining

$$\tilde{\gamma}[b_k] \coloneqq \frac{1}{n} \sum_{j=0}^{\Delta_k} \alpha[b_{k+1} - j] \sum_{i=1}^{n} \delta_i[b_{k+1} - j](\nabla f_i(\overline{x}[b_{k+1} - 1 - j]) - \nabla f_i(z_i[b_{k+1} - j])),$$

$$\zeta[b_k] \coloneqq \frac{1}{n} \sum_{j=0}^{\Delta_k - 1} \alpha[b_{k+1} - j] \left(- \sum_{i=1}^{n} \delta_i[b_{k+1} - j] \nabla f_i(\overline{x}[b_{k+1} - 1 - j]) \right),$$

which represent the consensus error and asynchrony error respectively, where $\{b_k\}$ *is the subsequence defined in Remark 4,* $\Delta_k \coloneqq b_{k+1} - b_k - 1$ *is related to the number of time indices between successive subsequence terms, and* $d[b_k] \coloneqq -\sum_{i=1}^n \nabla f_i(\overline{x}[b_k])$ *is the negative gradient of the global objective* $\overline{F}(\cdot)$ *evaluated at* $\overline{x}[b_k]$*, then it holds that*

$$\begin{aligned}\|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 \\ + \frac{2}{n}\langle \alpha[b_k+1]d[b_k] + \tilde{\gamma}[b_k] + \zeta[b_k], \overline{x}[b_k] - x^\star\rangle \\ + (\overline{\tau}^{proc})^2(\alpha[b_k+1])^2L^2.\end{aligned}$$

Proof. Recall the update equation (5.4) given by

$$\begin{aligned}\boldsymbol{x}[k+1] &= \boldsymbol{w}[k] - \alpha[k+1]\nabla\overline{\boldsymbol{F}}[k+1] \\ &= \overline{\boldsymbol{P}}[k]\boldsymbol{x}[k] - \alpha[k+1]\nabla\overline{\boldsymbol{F}}[k+1].\end{aligned}$$

Since the $\overline{\boldsymbol{P}}[k]$ are column stochastic, we can multiply each side of (5.4) by $\mathbf{1}^T/n$ and obtain

$$\overline{x}[k+1] = \overline{x}[k] - \frac{\alpha[k+1]}{n}\sum_{i=1}^n \delta_i[k+1]\nabla f_i(z_i[k+1]). \tag{5.12}$$

Adding and subtracting $\frac{\alpha[k+1]}{n}\sum_{i=1}^n \delta_i[k+1]\nabla f_i(\overline{x}[k])$ in (5.12) gives

$$\begin{aligned}\overline{x}[k+1] =& \overline{x}[k] - \frac{\alpha[k+1]}{n}\sum_{i=1}^n \delta_i[k+1]\nabla f_i(\overline{x}[k]) \\ &+ \frac{\alpha[k+1]}{n}\sum_{i=1}^n \delta_i[k+1](\nabla f_i(\overline{x}[k] - \nabla f_i(z_i[k+1])).\end{aligned} \tag{5.13}$$

Let $\gamma[k+1] \coloneqq \sum_{i=1}^{n} \delta_i[k+1](\nabla f_i(\overline{x}[k]) - \nabla f_i(z_i[k+1]))$. Hence we can rewrite (5.13) as

$$\overline{x}[k+1] = \overline{x}[k] - \frac{\alpha[k+1]}{n} \sum_{i=1}^{n} \delta_i[k+1] \nabla f_i(\overline{x}[k]) + \frac{\alpha[k+1]}{n} \gamma[k+1].$$

Recursing for some $\Delta_k \leq \overline{\tau}^{\text{proc}} - 1$ time indices until we find an iteration at which the agents' computations partially overlap (cf. Remark 4), and, without any loss of generality, let $k+1$ be an iteration at which agents' computations partially overlap, we have

$$\begin{aligned} \overline{x}[b_{k+1}] =& \overline{x}[b_k] - \frac{\alpha[b_k+1]}{n} \sum_{i=1}^{n} \nabla f_i(\overline{x}[b_k]) \\ &+ \sum_{j=0}^{\Delta_k} \frac{\alpha[k+1-j]}{n} \gamma[k+1-j] \\ &+ \sum_{j=0}^{\Delta_k - 1} \frac{\alpha[k+1-j]}{n} \left(- \sum_{i=1}^{n} \delta_i[k+1-j] \nabla f_i(\overline{x}[k-j]) \right). \end{aligned} \tag{5.14}$$

Let $d[b_k] \coloneqq -\sum_{i=1}^{n} \nabla f_i(\overline{x}[b_k])$, which is the negative gradient of the global objective $\overline{F}(\cdot)$ evaluated at $\overline{x}[b_k]$. In addition, let $\tilde{\gamma}[b_k] \coloneqq \frac{1}{n} \sum_{j=0}^{\Delta_k} \alpha[k+1-j]\gamma[k+1-j]$, which represents the "consensus error" in the update, and let $\zeta[b_k] \coloneqq \frac{1}{n} \sum_{j=0}^{\Delta_k - 1} \alpha[k+1-j] \left(- \sum_{i=1}^{n} \delta_i[k+1-j] \nabla f_i(\overline{x}[k-j]) \right)$, which represents the "asynchrony error" in the update. Equation (5.14) simplifies to

$$\overline{x}[b_{k+1}] = \overline{x}[b_k] + \frac{1}{n} \alpha[b_k+1] d[b_k] + \tilde{\gamma}[b_k] + \zeta[b_k]. \tag{5.15}$$

Subtracting $x^\star$ from each side of (5.15) and taking the squared norm

$$\begin{aligned}\|\overline{x}[b_{k+1}] - x^\star\|^2 = &\|\overline{x}[b_k] - x^\star\|^2 \\ &+ 2\langle \frac{1}{n}\alpha[b_k+1]d[b_k] + \tilde{\gamma}[b_k] + \zeta[b_k], \overline{x}[b_k] - x^\star\rangle \\ &+ \left\| \frac{1}{n}\alpha[b_k+1]d[b_k] + \tilde{\gamma}[b_k] + \zeta[b_k] \right\|^2 .\end{aligned} \tag{5.16}$$

Note that

$$\frac{1}{n}\alpha[b_k+1]d[b_k] + \tilde{\gamma}[b_k] + \zeta[b_k] = \sum_{j=0}^{\Delta_k} \frac{1}{n}\alpha[k+1-j] \left(-\sum_{i=1}^{n} \delta_i[k+1-j]\nabla f_i(z_i[k+1-j]) \right).$$

Recalling the assumption that the step-size sequence is non-increasing, and noting that $\Delta_k < \overline{\tau}^{\text{proc}}$ and $\delta_i[k] \in \{0,1\}$, we can use the triangle inequality to obtain

$$\left\| \frac{1}{n}\alpha[b_k+1]d[b_k] + \tilde{\gamma}[b_k] + \zeta[b_k] \right\|^2 \leq (\overline{\tau}^{\text{proc}})^2 (\frac{1}{n}\alpha[b_k+1])^2 n^2 L^2. \tag{5.17}$$

Substituting (5.17) into the update equation (5.16) gives the desired result

$$\begin{aligned}\|\overline{x}[b_{k+1}] - x^\star\|^2 \leq &\|\overline{x}[b_k] - x^\star\|^2 \\ &+ 2\langle \frac{1}{n}\alpha[b_k+1]d[b_k] + \tilde{\gamma}[b_k] + \zeta[b_k], \overline{x}[b_k] - x^\star\rangle \\ &+ (\overline{\tau}^{\text{proc}})^2 (\alpha[b_k+1])^2 L^2.\end{aligned}$$

■

Lemma 5.4. *If Assumption 1 and Assumption 2 are satisfied, and the step-size sequence $\{\alpha[k]\}$ is non-increasing and satisfies*

$$\alpha[0] \leq \left(\frac{m}{2M^2}\right) \left(\frac{1}{N_{max}^{out}}\right)^{n(\overline{\tau}+1)},$$

then

$$\langle \tilde{\gamma}[b_k], \overline{x}[b_k] - x^\star \rangle \leq (\overline{\tau}^{proc}) M\sqrt{d}\frac{L}{m}\alpha[b_k+1]\left(C\left\|x_0\right\| q^{b_k-\overline{\tau}^{proc}} + CL\sum_{s=0}^{b_k} q^{b_k-s}\alpha[s]\right),$$

$$\langle \zeta[b_k], \overline{x}[b_k] - x^\star \rangle \leq \frac{(n-1)}{n}(\overline{\tau}^{proc}-1)\alpha[b_k+1]\left(L\left\|x^\star - \overline{x}[b_k]\right\| - \frac{m}{2}\left\|x^\star - \overline{x}[b_k]\right\|^2\right),$$

where $\tilde{\gamma}[b_k]$ and $\zeta[b_k]$ are the consensus and asynchrony error terms, respectively, defined in Lemma 5.3, q and C are the constants defined in Theorem 1, and D and L are the bounds constructed in Theorem 5.

Proof. We can bound the consensus error inner product, $\langle \tilde{\gamma}[b_k], \overline{x}[b_k] - x^\star \rangle$, as follows:

$$\begin{aligned}
\langle \tilde{\gamma}[b_k], \overline{x}[b_k] - x^\star \rangle \coloneqq & \frac{1}{n}\sum_{j=0}^{\Delta_k}\alpha[k+1-j]\Bigg(\sum_{i=1}^{n}\delta_i[k+1]\langle \nabla f_i(\overline{x}[k-j]) \\
& - \nabla f_i(z_i[k+1-j]), \overline{x}[b_k] - x^\star\rangle\Bigg) \\
\leq & (\overline{\tau}^{\text{proc}})\alpha[b_k+1]\max_{i,j}\left\|\nabla f_i(\overline{x}[k-j]) - \nabla f_i(z_i[k+1-j])\right\|_1 \left\|\overline{x}[b_k] - x^\star\right\|_\infty \\
\leq & (\overline{\tau}^{\text{proc}})M\sqrt{d}\frac{L}{m}\alpha[b_k+1]\max_{i,j}\left\|z_i[k+1-j] - \overline{x}[k-j]\right\| \\
\leq & (\overline{\tau}^{\text{proc}})M\sqrt{d}\frac{L}{m}\alpha[b_k+1]\left(C\left\|x_0\right\| q^{b_k-\overline{\tau}^{\text{proc}}} + CL\sum_{s=0}^{b_k} q^{b_k-s}\alpha[s]\right).
\end{aligned}$$

Here we have explicitly used the gradient Lipschitz-continuity; the consensus rate bound from Theorem 1; the strong-convexity of the objective; and the fact that the step-size sequence is non-increasing.

Now we bound the asynchrony error inner product $\langle \zeta[b_k], \overline{x}[b_k] - x^\star \rangle$; by substituting in the definition for $\zeta[b_k]$ we have that

$$\langle \zeta[b_k], \overline{x}[b_k] - x^\star \rangle \coloneqq \sum_{j=0}^{\Delta_k - 1}\tilde{\alpha}[k+1-j]\left(\sum_{i=1}^{n}\delta_i[k+1-j]\langle \nabla f_i(\overline{x}[k-j]), x^\star - \overline{x}[b_k]\rangle\right).$$

Bounding each one of the terms in the sum individually, we can say that

$$\langle \nabla f_i(\overline{x}[k-j]), x^\star - \overline{x}[b_k] \rangle \leq f_i(x^\star - \overline{x}[b_k] + \overline{x}[k-j]) - f_i(\overline{x}[k-j]) - \frac{m}{2} \left\| x^\star - \overline{x}[b_k] \right\|^2$$
$$\leq L \left\| x^\star - \overline{x}[b_k] \right\| - \frac{m}{2} \left\| x^\star - \overline{x}[b_k] \right\|^2 .$$

Now by making use of the fact that $\Delta_k \leq \overline{\tau}^{\text{proc}} - 1$, and recalling that the step-size sequence is non-increasing, we can substitute these term-by-term bounds back into the inner product relation to obtain

$$\langle \zeta[b_k], \overline{x}[b_k] - x^\star \rangle \leq \frac{(n-1)}{n} (\overline{\tau}^{\text{proc}} - 1) \alpha[b_k + 1] \left(L \left\| x^\star - \overline{x}[b_k] \right\| - \frac{m}{2} \left\| x^\star - \overline{x}[b_k] \right\|^2 \right),$$

the desired result. ■

5.3.2 Proof of Theorem 2

Proof of Theorem 2. Define the lim inf sequence $\{m[k]\}$ given by $m[k+1] \coloneqq \min\{ \overline{F}(\overline{x}[k]), m[k] \}$. Note that $\{m[k]\}$ is monotonically decreasing and bounded below by $\overline{F}(x^\star)$, hence the sequence converges to some $m^\star$. *i.e.*, $m[k] \downarrow m^\star \geq \overline{F}(x^\star)$. Let $\rho \coloneqq (n-1)(\overline{\tau}^{\text{proc}} - 1) L/m$, and define

$$v[b_k] \coloneqq x^\star - \rho \frac{d[b_k]}{\| d[b_k] \|}.$$

Let $\beta \coloneqq F(x^\star) + L\rho$, and assume for the sake of a contradiction that $m^\star \geq \beta > \overline{F}(x^\star)$. From the Lipschitz-continuity of the global objective implied by the gradient bound in Theorem 5, we have

$$F(v[b_k]) - F(x^\star) \leq L \left\| v[b_k] - x^\star \right\| = L\rho,$$

hence

$$F(v[b_k]) \leq F(x^\star) + L\rho = \beta.$$

That is, $v[b_k]$ is in the β-sublevel set of $F(\cdot)$. Recalling that $d[b_k]$ is the negative gradient of the global objective and using the first-order definition of convexity we have

$$\langle v[b_k] - \overline{x}[b_k], -d[b_k] \rangle \leq \overline{F}(v[b_k]) - \overline{F}(\overline{x}[b_k]) \leq \beta - m[b_k + 1] \leq 0. \tag{5.18}$$

Substituting in the definition of $v[b_k]$ into (5.18) gives

$$\langle x^\star - \overline{x}[b_k] - \rho \frac{d[b_k]}{\|d[b_k]\|}, -d[b_k] \rangle = \langle \overline{x}[b_k] - x^\star, d[b_k] \rangle + \rho \|d[b_k]\| \leq 0,$$

therefore

$$\langle \overline{x}[b_k] - x^\star, d[b_k] \rangle \leq -\rho \|d[b_k]\| < 0. \tag{5.19}$$

Substituting (5.19) into the result of Lemma 5.3 gives the following update relation:

$$\begin{aligned} \|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 & \\ + 2\langle \tilde{\gamma}[b_k] + \zeta[b_k], \overline{x}[b_k] - x^\star \rangle & \\ + (\overline{\tau}^{\text{proc}})^2 (\alpha[b_k + 1])^2 L^2 & \\ - 2\frac{1}{n} \alpha[b_k + 1] \rho \|d[b_k]\|. & \end{aligned} \tag{5.20}$$

Applying the bounds from Lemma 5.4 directly, and defining the constants

$$\begin{aligned}
C_1 &\coloneqq (n-1)(\overline{\tau}^{\text{proc}} - 1), \\
C_2 &\coloneqq 2(\overline{\tau}^{\text{proc}})M\sqrt{d}\frac{L}{m}C \left\|x_0\right\| q^{-\overline{\tau}^{\text{proc}}}, \\
C_3 &\coloneqq 2(\overline{\tau}^{\text{proc}})M\sqrt{d}\frac{L^2}{m}C, \\
C_4[b_k] &\coloneqq L \left\|x^\star - \overline{x}[b_k]\right\| - \frac{m}{2}\left\|x^\star - \overline{x}[b_k]\right\|^2,
\end{aligned}$$

the update relation (5.20) simplifies to

$$\begin{aligned}
\left\|\overline{x}[b_{k+1}] - x^\star\right\|^2 \leq \left\|\overline{x}[b_k] - x^\star\right\|^2 & \\
&+ (\overline{\tau}^{\text{proc}} L\alpha[b_k+1])^2 \\
&- 2\rho\frac{1}{n}\left\|d[b_k]\right\|\alpha[b_k+1] \\
&+ C_2\alpha[b_k+1]q^{b_k} + C_3\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s] \\
&+ 2\frac{C_1 C_4[b_k]}{n}\alpha[b_k+1].
\end{aligned}$$

Grouping terms together gives

$$\begin{aligned}
\left\|\overline{x}[b_{k+1}] - x^\star\right\|^2 \leq \left\|\overline{x}[b_k] - x^\star\right\|^2 & \\
&+ \alpha[b_k+1](\alpha[b_k+1](\overline{\tau}^{\text{proc}}L)^2 - \frac{2}{n}(\rho\left\|d[b_k]\right\| - C_1C_4[b_k])) \\
&+ C_2\alpha[b_k+1]q^{b_k} \\
&+ C_3\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s].
\end{aligned} \tag{5.21}$$

Now by invoking the strong convexity of $F(\cdot)$ and noting that $\nabla F(x^\star) = 0$ (since, by definition, $x^\star$ is the global optimum), we have

$$\begin{aligned}\frac{m}{2}\left\|\overline{x} - x^\star\right\|^2 \leq F(\overline{x}) - F(x^\star) &\leq \langle \nabla F(\overline{x})^\top, \overline{x} - x^\star \rangle - \frac{m}{2}\left\|\overline{x} - x^\star\right\|^2, \\ &\leq \left\|\nabla F(\overline{x})\right\| \left\|\overline{x} - x^\star\right\| - \frac{m}{2}\left\|\overline{x} - x^\star\right\|^2,\end{aligned}$$

which implies that

$$m\left\|\overline{x} - x^\star\right\| \leq \left\|\nabla F(\overline{x})\right\|. \tag{5.22}$$

Using (5.22), which holds in general for all strongly-convex functions, and applying the definition of $\rho \coloneqq (n-1)(\overline{\tau}^{\text{proc}} - 1)L/m$, observe that

$$(n-1)(\overline{\tau}^{\text{proc}} - 1)L\left\|\overline{x}[b_k] - x^\star\right\| \leq \rho\left\|d[b_k]\right\|,$$

hence

$$\rho\left\|d[b_k]\right\| - (n-1)(\overline{\tau}^{\text{proc}} - 1)C_4[b_k] \geq \epsilon > 0 \tag{5.23}$$

for all b_k and a scalar $\epsilon > 0$. Since the step-size is decreasing, it follows that there exists an index b_r such that for all $b_k \geq b_r$

$$\alpha[b_k] \leq \epsilon \leq \frac{\rho\left\|d[b_k]\right\| - C_1 C_4[b_k]}{n(\overline{\tau}^{\text{proc}} L)^2}. \tag{5.24}$$

Therefore, by applying (5.24) to (5.21) it holds that for all $b_k \geq b_r$ that

$$\begin{aligned}\|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 & \\ - \frac{\alpha[b_k+1]}{n}(\rho\,\|d[b_k]\| - C_1 C_4[b_k]) & \\ + C_2\alpha[b_k+1]q^{b_k} & \\ + C_3\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s], &\end{aligned}$$

which implies

$$\begin{aligned}\frac{\epsilon}{n}\alpha[b_k+1] \leq \|\overline{x}[b_k] - x^\star\|^2 - \|\overline{x}[b_{k+1}] - x^\star\|^2 & \\ + C_2\alpha[b_k+1]q^{b_k} & \\ + C_3\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s], &\end{aligned}$$

for all $b_k \geq b_r$. Summing over the subsequence, and noticing that we have a telescoping sum on the right hand side, gives

$$\begin{aligned}\frac{\epsilon}{n}\sum_{k=r}^{\ell}\tilde{\alpha}[b_k+1] \leq & \|\overline{x}[b_r]-x^\star\|^2 - \|\overline{x}[b_\ell]-x^\star\|^2 \\ & + C_2\alpha[b_r+1]\frac{1}{1-q} \\ & + C_3\sum_{k=r}^{\ell}\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s], \\ \leq & \|\overline{x}[b_r]-x^\star\|^2 \\ & + C_2\alpha[b_r+1]\frac{1}{1-q} \\ & + C_3\sum_{k=r}^{\ell}\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s].\end{aligned}$$

Since $q \in (0,1)$, and by assumption $\sum_{s=0}^{\infty}\alpha^2[s] < \infty$, it follows from [67, Lemma 3.1] that

$$\sum_{k=0}^{\infty}\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s] < \infty,$$

hence

$$\begin{aligned}\frac{\epsilon}{n}\sum_{k=r}^{\infty}\alpha[b_k+1] \leq & \|\overline{x}[b_r]-x^\star\|^2 \\ & + C_2\alpha[b_r+1]\frac{1}{1-q} \\ & + C_3\sum_{k=r}^{\infty}\sum_{s=0}^{b_k} q^{b_k-s}\alpha^2[s] \\ < & \infty.\end{aligned} \tag{5.25}$$

This is a contradiction since the step-size sum on the left hand side of (5.25) sums to infinity. Hence $m^\star < \beta$. Therefore

$$\liminf_{k\to\infty} \frac{m}{2} \|\overline{x}[k] - x^\star\|^2 \leq \liminf_{k\to\infty} F(\overline{x}[k]) - F(x^\star) = m^\star - F(x^\star) \tag{5.26}$$

$$< \beta - F(x^\star) = L\rho = \frac{(n-1)(\overline{\tau}^{\text{proc}} - 1)L^2}{m}, \tag{5.27}$$

where (5.26) simply follows from the strong convexity of the global objective, and the (5.27) is a result of the contradiction. Hence,

$$\liminf_{k\to\infty} \|\overline{x}[k] - x^\star\| < \frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)}.$$

Since for all $i = 1, 2, \ldots, n$, the iterates $z_i[k+1]$ converge to $\overline{x}[k]$, we add an subtract $z_i[k+1]$ to obtain

$$\begin{aligned}
\frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)} &> \liminf_{k\to\infty} \|z_i[k+1] + \overline{x}[k] - z_i[k+1] - x^\star\|, \\
&\geq \liminf_{k\to\infty} \left(\|z_i[k+1] - x^\star\| - \|\overline{x}[k] - z_i[k+1]\|\right), \\
&= \liminf_{k\to\infty} \|z_i[k+1] - x^\star\|,
\end{aligned}$$

where the last equality is due to Corollary 1.2. ■

5.3.3 Proof of Theorem 3

Proof of Theorem 3. Define the lim inf sequence $\{m[k]\}$ given by $m[k+1] \coloneqq \min\{\overline{F}(\overline{x}[k]), m[k]\}$. Note that $\{m[k]\}$ is monotonically decreasing and bounded below by $\overline{F}(x^\star)$, hence the sequence converges to some $m^\star$. *i.e.*, $m[k] \downarrow m^\star \geq \overline{F}(x^\star)$. Let

$\rho \coloneqq (n-1)(\overline{\tau}^{\text{proc}} - 1)L/m$, and define

$$v[b_k] \coloneqq x^\star - \rho \frac{d[b_k]}{\|d[b_k]\|}. \tag{5.28}$$

Let $\beta \coloneqq F(x^\star) + L\rho$, and assume for the sake of a contradiction that $m^\star \geq \beta > \overline{F}(x^\star)$. The proof is identical to that of Theorem 2 up to (5.19), which tell us that

$$\begin{aligned} \|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 \\ + 2\langle \tilde{\gamma}[b_k] + \zeta[b_k], \overline{x}[b_k] - x^\star \rangle \\ + (\overline{\tau}^{\text{proc}})^2 \alpha^2 L^2 \\ - \frac{2}{n} \alpha\rho \|d[b_k]\|. \end{aligned} \tag{5.29}$$

Applying the bounds from Lemma 5.4 directly, and defining the constants

$$\begin{aligned} C_1 &\coloneqq ((n-1)(\overline{\tau}^{\text{proc}} - 1))^2, \\ C_2 &\coloneqq 2(\overline{\tau}^{\text{proc}}) M\sqrt{d} \frac{L}{m} C \|x_0\| q^{-\overline{\tau}^{\text{proc}}}, \\ C_3 &\coloneqq 2(\overline{\tau}^{\text{proc}}) M\sqrt{d} \frac{L^2}{m} C \frac{1}{1-q}, \\ C_4[b_k] &\coloneqq L \|x^\star - \overline{x}[b_k]\| - \frac{m}{2} \|x^\star - \overline{x}[b_k]\|^2, \end{aligned}$$

the update relation (5.29) simplifies to

$$\begin{aligned}\|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 & \\ + (\overline{\tau}^{\text{proc}} L\alpha)^2 & \\ - \frac{2}{n}\rho \|d[b_k]\| \alpha & \\ + C_2 \alpha q^{b_k} + C_3\alpha^2 & \\ + \frac{2C_1 C_4[b_k]}{n}\alpha. &\end{aligned}$$

Grouping terms together gives

$$\begin{aligned}\|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 & \\ + \alpha(\alpha((\overline{\tau}^{\text{proc}} L)^2 + C_3) - \frac{2}{n}(\rho \|d[b_k]\| - C_1 C_4[b_k])) & \\ + C_2 \alpha q^{b_k}. &\end{aligned}$$

Using (5.22), which holds for all strongly convex functions, and applying the definition of $\rho \coloneqq (n-1)(\overline{\tau}^{\text{proc}} - 1)L/m$, observe that

$$(n-1)(\overline{\tau}^{\text{proc}} - 1)L \|\overline{x}[b_k] - x^\star\| \leq \rho \|d[b_k]\|,$$

hence $\rho \|d[b_k]\| - (n-1)(\overline{\tau}^{\text{proc}} - 1)C_4[b_k] \geq \epsilon > 0$ for all b_k and a scalar $\epsilon > 0$. Now going back to our assumption

$$\liminf_{k\to\infty} \overline{F}(\overline{x}[k]) \coloneqq m^\star \geq \beta \coloneqq F(x^\star) + L\rho,$$

we see that this implies

$$\begin{aligned} L\rho &\leq \liminf_{k\to\infty} \overline{F}(\overline{x}[k]) - F(x^\star) \\ &\leq L \liminf_{k\to\infty} \|\overline{x}[k] - x^\star\|, \end{aligned}$$

and therefore,

$$\rho \leq \|\overline{x}[k] - x^\star\|, \tag{5.30}$$

for all k. We now make use of this result. Since, by assumption, the step-size satisfies

$$\alpha \leq \frac{((n-1)(\overline{\tau}^{\text{proc}}-1))^3 L^2}{2nm((\overline{\tau}^{\text{proc}}L)^2 + C_3)} = \frac{m(n-1)(\overline{\tau}^{\text{proc}}-1)}{2n((\overline{\tau}^{\text{proc}}L)^2 + C_3)}\rho^2,$$

it follows that the step-size also satisfies

$$\begin{aligned} \alpha &\leq \frac{1}{n((\overline{\tau}^{\text{proc}}L)^2 + C_3)} \left(\frac{m(n-1)(\overline{\tau}^{\text{proc}}-1)}{2} \|x^\star - \overline{x}[b_k]\|^2 \right), \\ &= \frac{1}{n((\overline{\tau}^{\text{proc}}L)^2 + C_3)} \left(\rho \|d[b_k]\| - C_1 \frac{L}{m} \|d[b_k]\| + \frac{mC_1}{2} \|x^\star - \overline{x}[b_k]\|^2 \right), \\ &\leq \frac{1}{n((\overline{\tau}^{\text{proc}}L)^2 + C_3)} \left(\rho \|d[b_k]\| - C_1 \left(L \|x^\star - \overline{x}[b_k]\| - \frac{m}{2} \|x^\star - \overline{x}[b_k]\|^2 \right) \right) \\ &= \frac{1}{n((\overline{\tau}^{\text{proc}}L)^2 + C_3)} \left(\rho \|d[b_k]\| - C_1 C_4[b_k] \right). \end{aligned}$$

Therefore,

$$\begin{aligned} \|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 & \\ - \frac{\alpha}{n}(\rho \|d[b_k]\| - (n-1)(\overline{\tau}^{\text{proc}}-1)C_4[b_k]) & \\ + C_2 \alpha q^{b_k}, & \end{aligned}$$

which implies

$$\frac{(\rho \|d[b_k]\| - (n-1)(\overline{\tau}^{\text{proc}} - 1)C_4[b_k])}{n}\alpha \leq \|\overline{x}[b_k] - x^\star\|^2 - \|\overline{x}[b_{k+1}] - x^\star\|^2 + C_2\alpha q^{b_k}. \tag{5.31}$$

Summing (5.31) over the subsequence, using (5.23), and noticing that we have a telescoping sum on the right hand side, gives

$$\begin{aligned}\frac{\epsilon}{n}\sum_{k=0}^{\ell}\alpha &\leq \|\overline{x}[b_0] - x^\star\|^2 - \|\overline{x}[b_\ell] - x^\star\|^2 \\ &\quad + C_2\alpha\frac{1}{1-q}, \\ &\leq \|\overline{x}[b_0] - x^\star\|^2 + C_2\alpha\frac{1}{1-q},\end{aligned}$$

hence

$$\begin{aligned}\frac{\epsilon}{n}\sum_{k=0}^{\infty}\alpha &\leq \|\overline{x}[b_0] - x^\star\|^2 \\ &\quad + C_2\alpha\frac{1}{1-q} \\ &< \infty.\end{aligned}$$

This is a contradiction since the step-size sum on the left hand side of the inequality sums to infinity. Hence $m^\star < \beta$. Therefore

$$\begin{aligned}\liminf_{k\to\infty}\frac{m}{2}\|\overline{x}[k] - x^\star\|^2 &\leq \liminf_{k\to\infty} F(\overline{x}[k]) - F(x^\star) = m^\star - F(x^\star) \\ &< \beta - F(x^\star) = L\rho = \frac{(n-1)(\overline{\tau}^{\text{proc}} - 1)L^2}{m},\end{aligned}$$

where the first inequality simply follows from the strong convexity of the global objective, and the second inequality is a result of the contradiction. Hence

$$\liminf_{k\to\infty} \|\overline{x}[k] - x^\star\| < \frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)}.$$

Since for all $i \in \{1, \ldots, n\}$ the iterates $z_i[k+1]$ converge to a neighbourhood of $\overline{x}[k]$, we add an subtract $z_i[k+1]$ to obtain

$$\begin{aligned}
\frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)} &> \liminf_{k\to\infty} \|z_i[k+1] + \overline{x}[k] - z_i[k+1] - x^\star\|, \\
&\geq \liminf_{k\to\infty} (\|z_i[k+1] - x^\star\| - \|\overline{x}[k] - z_i[k+1]\|) \\
&\geq \liminf_{k\to\infty} \|z_i[k+1] - x^\star\| - \liminf_{k\to\infty} \|\overline{x}[k] - z_i[k+1]\|.
\end{aligned}$$

Therefore

$$\begin{aligned}
\liminf_{k\to\infty} \|z_i[k+1] - x^\star\| &< \liminf_{k\to\infty} \|\overline{x}[k] - z_i[k+1]\| + \frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)} \\
&\leq \alpha\frac{\widetilde{C}L}{1-q} + \frac{L}{m}\sqrt{2(\overline{\tau}^{\text{proc}} - 1)(n-1)}.
\end{aligned}$$

where the last inequality follows from Corollary 1.1 ■

5.3.4 Proof of Theorem 4

The proof is also identical to that of Theorem 2 up to (5.20), which tells us that

$$\begin{aligned}
\|\overline{x}[b_{k+1}] - x^\star\|^2 \leq \|\overline{x}[b_k] - x^\star\|^2 + (\overline{\tau}^{\text{proc}}\alpha L)^2 + C_2\alpha q^{b_k} \\
- \frac{2\rho\,\|d[b_k]\|}{n}\alpha + \frac{C_3\alpha^2}{1-q},
\end{aligned}$$

where here we have arbitrarily defined $\rho > 0$, and assumed, for the sake of a contradiction, that

$$\liminf_{k\to\infty} \overline{F}(\overline{x}[k]) \coloneqq m^\star \geq \beta \coloneqq F(x^\star) + L\rho.$$

Since, by assumption, the step-size satisfies

$$\alpha \leq \frac{m\rho^2}{n((\overline{\tau}^{\mathrm{proc}}L)^2 + \frac{C_3}{1-q})},$$

it follows from (5.30) and (5.22) that the step-size also satisfies

$$\alpha \leq \frac{\rho \left\| d[b_k] \right\|}{n((\overline{\tau}^{\mathrm{proc}}L)^2 + \frac{C_3}{1-q})}. \tag{5.32}$$

Therefore, using (5.32), we have

$$\frac{m\rho^2}{n}\alpha \leq \left\| \overline{x}[b_k] - x^\star \right\|^2 - \left\| \overline{x}[b_{k+1}] - x^\star \right\|^2 + C_2\alpha q^{b_k}. \tag{5.33}$$

Summing (5.33) over the subsequence and noticing that we have a telescoping sum on the right hand side, gives

$$\frac{m\rho^2}{n}\sum_{k=0}^{\ell} \alpha \leq \left\| \overline{x}[b_0] - x^\star \right\|^2 + \alpha\frac{C_2}{1-q} < \infty.$$

This is a contradiction; hence $m^\star < \beta$, and therefore

$$\liminf_{k\to\infty} \frac{m}{2} \left\| \overline{x}[k] - x^\star \right\|^2 \leq \liminf_{k\to\infty} F(\overline{x}[k]) - F(x^\star) \tag{5.34}$$

$$< \beta - F(x^\star) = L\rho, \tag{5.35}$$

where (5.34) follows from the strong convexity of the global objective, and (5.35) is a result of the contradiction. Making use of (5.35) and Corollary 1.1, it follows that

$$\liminf_{k\to\infty} \|z_i[k+1] - x^\star\| < \frac{\alpha C L}{1-q} + \sqrt{\frac{2L\rho}{m}}.$$

■

5.4 Numerical Experiments

We report experiments on a high performance computing cluster. The cluster makes use of a QDR InfiniBand network capable of 40 Gbps to each node and hosts large scale storage systems operating with an optimized parallel file system. The Intel-MPI distribution is used with Python bindings (mpi4py) for message passing. The communication graphs (network topologies) are randomly generated; amongst other parameters, our implementation of the graph generation algorithm allows us to specify the desired size of the graph, as well as the average out-degree of each agent. The generator produces a number of graphs and confirms that they are strongly connected by performing a breadth-first search through the graphs. Subsequently, the generator takes the adjacency matrices corresponding to the strongly-connected graphs and scores them in terms of information diffusion speed. This is done by creating an equivalent doubly stochastic uniform edge weighted matrix, using a procedure similar to that proposed in [13], and by subsequently computing the second largest eigenvalue of this matrix (the largest is equal to 1 (cf. [8])). The graph with the smallest spectral radius is saved. Table 5–1 shows the spectral radii of the uniform edge weighted matrices corresponding to the generated graphs.

Table 5–1: Spectral radii of generated multi-agent networks. Larger spectral-radii indicate more sparsely connected graphs. The spectral radius is computed as $\frac{1}{1-\lambda_2}$, where λ_2 is the second-largest eigenvalue of the uniform edge weighted matrix used to score the graphs. For fully-connected graphs, λ_2 is equal to 0. As the graph connections are made more and more sparse, the second-largest eigenvalue will approach 1 from below: $\lambda_2 \uparrow 1$.

Num. agents	**Spectral radius**
2	1.000
4	1.000
8	1.498
16	2.500
32	4.121
64	6.447

A multinomial logistic regression classifier (softmax predictor) is trained on the *Covertype* dataset [45] (available from the *UCI repository*) using the negative log-likelihood loss function:

$$\min_{w \in \mathbb{R}^d} F(\boldsymbol{X}, \boldsymbol{y}|\boldsymbol{w}) \coloneqq -\sum_{i=1}^{D}\sum_{j=1}^{K} log\left(\frac{\exp(w_j^T x^i)}{\sum_{j'=1}^{K}\exp(w_{j'}^T x^i)}\right)^{y_j^i},$$

where $D \in \mathbb{R}_{++}$ is the number of training instances in the dataset, $K \in \mathbb{R}_{++}$ is the number of classes, $x^i \in \mathbb{R}^d$ and $y^i \in \mathbb{R}^K$ correspond to the i^{th} training instance feature and label vectors respectively (the label vectors are represented using a 1-hot encoding), and $w \in \mathbb{R}^d$ are the parameterizing weights. The dataset contains 581012 data samples, and 54 raw predictive features. Consequently, the optimizer solves for 378 (=54×7) parametrizing weights. The optimization problem is distributed by giving each agent in the multi-agent network a subset of the data samples from which to construct local negative log-likelihood loss functions, $f_i(w_i)$. The vector $w_i \in \mathbb{R}^{378}$ denotes the parameterizing weight vector held locally by agent v_i. The goal is to use

Table 5–2: Statistics concerning the time taken by agent v_1 to perform an update in the reported experiments for multiple different network sizes in the **Fixed Problem Workload** formulation.

Num. agents	Mean time (s)	Max. time (s)	Min. time (s)	Std. (s)
2	1.085	1.322	0.959	0.050
4	0.316	0.426	0.288	0.037
8	0.194	1.302	0.148	0.089
16	0.180	0.256	0.101	0.017
32	0.032	0.080	0.021	0.009
64	0.013	0.031	0.009	0.005

these features to predict the cover-type of a geographic area; there are 7 cover-types in total, and hence 7 possible classes from which to make a prediction — this is a classification problem. The 54 features consist of a mix of categorical (binary 1 or 0) features and real numbers. We whiten the non-categorical features by subtracting the mean and dividing by the standard deviation. Each of the Push DIGing (PD), Extra Push (EP), Synchronous Subgradient Push (Synch-SSP), and Asynchronous Subgradient Push (Asynch-SSP) algorithms are used to minimize the negative log-likelihood of the softmax function constructed form the *Covertype* dataset. All step-sizes are hand-optimized using a simple grid-search; coincidentally, the Synchronous Subgradient Push, Extra Push, Push DIGing, and Asynchronous Subgradient Push methods all use the same (constant) step-size.

Fixed Problem Workload

In each n-agent multi-agent network, each agent uses $\sim 581012/n$ data samples to construct its local negative log-likelihood loss functions. These data samples are

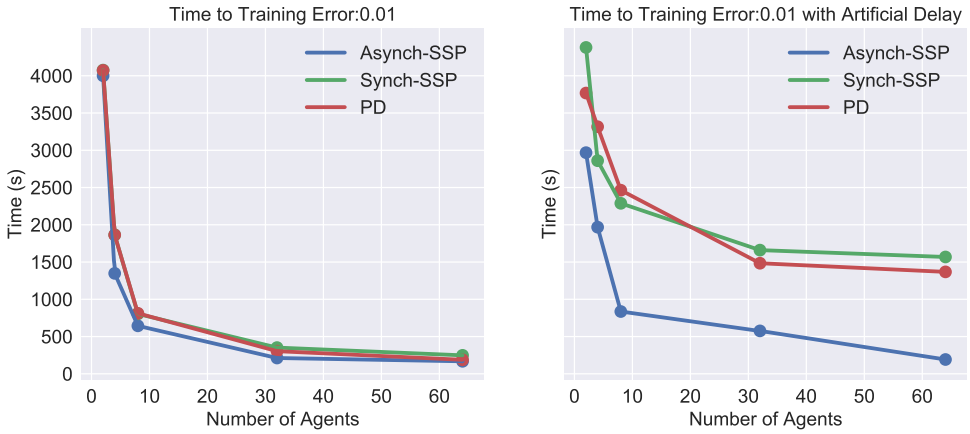
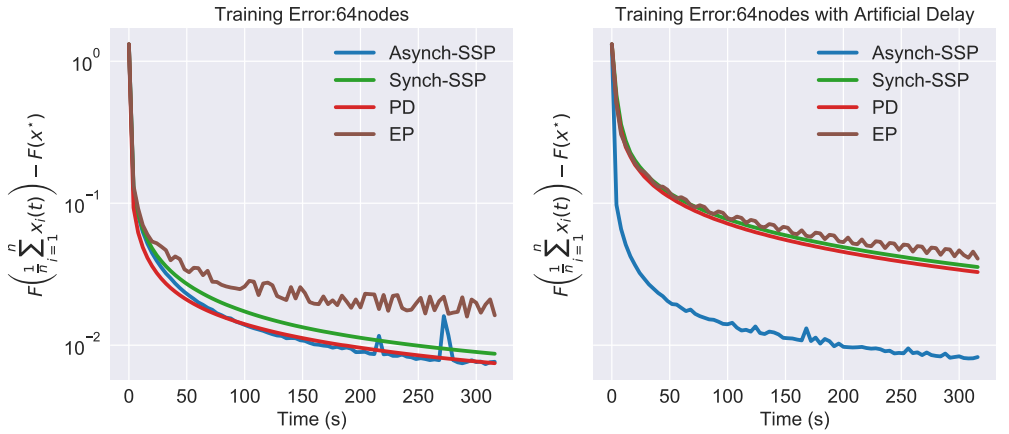


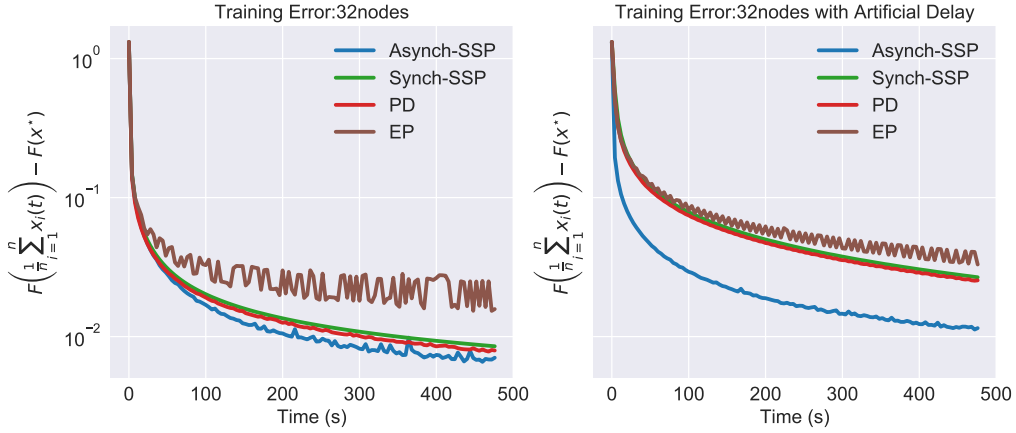
Figure 5–2: Time $t[k]$ (seconds) at which $F(\overline{x}[k]) - F(x^\star) < 0.01$ is satisfied for the first time. Plots on the right correspond to experiments with an artificial $500ms$ delay induced at agent v_2 at each of its local iterations. Plots on the left correspond to the normal operation of the algorithm. The asynchronous algorithm reaches the threshold residual error faster than the state-of-the-art methods. The Extra-Push algorithm is not plotted, because, in several cases, we were not able to find a step-size that enabled the method to achieve the target residual error in a reasonable amount of time; this is consistent with the observations in [58], where in some cases, there were no step-sizes that even lead to convergence.

non-overlapping, and so the problem at hand is considered to have a fixed computational workload; that is, as we increase the size of the multi-agent network, the computational load per agent decreases.

Discussion. Figure 5–3 shows how the individual algorithms scale with the network size as we keep the average out-degree of each agent fixed (*i.e.*, larger networks correspond to sparser topologies; cf. Table 5–1). Increasing the network size



(a)

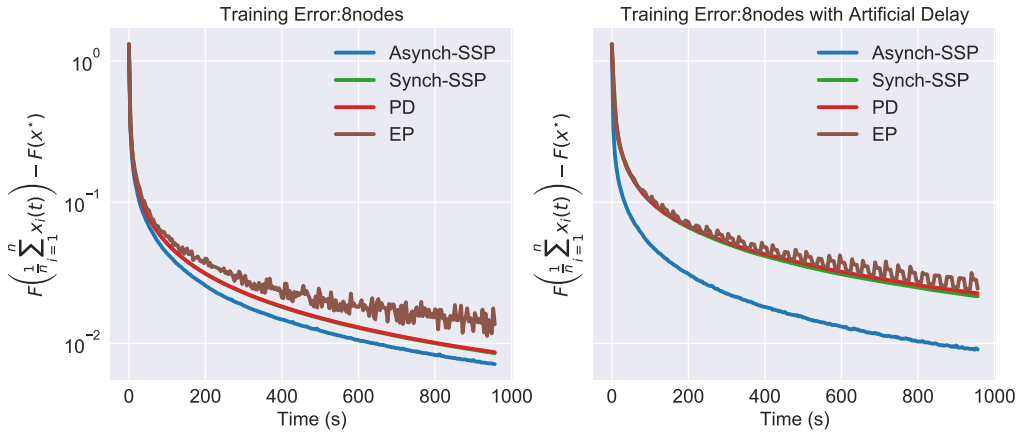


(b)

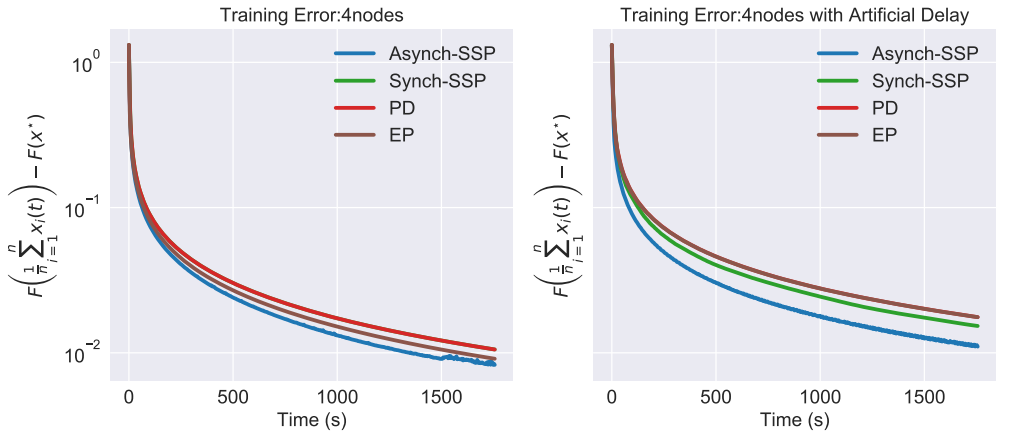
Figure 5–3: Multinomial logistic regression training error on the covertype dataset using large multi-agent networks. Plots on the right correspond to experiments with an artificial $500ms$ delay induced at agent v_2 at each of its local iterations. Plots on the left correspond to the normal operation of the algorithm. The asynchronous algorithm appears to be more robust than the synchronous algorithms to failing or stalling nodes.

appears to exhibit a sub-geometric improvement in the optimization time.[2] Figure 5–3 also shows that the asynchronous subgradient algorithm actually decreases

[2] The question of how to choose the optimal number of agents for a given distributed optimization problem is studied in depth in [73].



(a)



(b)

Figure 5–4: Multinomial logistic regression training error on the covertype dataset using small multi-agent networks. Plots on the right correspond to experiments with an artificial $500ms$ delay induced at agent v_2 at each of its local iterations. Plots on the left correspond to the normal operation of the algorithm. The asynchronous algorithm appears to be more robust than the synchronous algorithms to failing or stalling nodes.

the residual error for both small and large network sizes faster than the state-of-the art methods and its synchronous counterpart. This behaviour is even more

pronounced if one of the agents in the network works at a slower pace than the others. In particular, the asynchronous algorithm appears to be more robust than the synchronous algorithms to failing or stalling nodes. Figures 5–3 and 5–4 show the residual error of the algorithms for different network sizes with an artificial delay induced at agent v_2 at each iteration. The synchronous algorithms experience a significant slowdown relative to the asynchronous algorithm, which is much less affected. This observation is also clear from Figure 5–3, where the time to reach a threshold error is plotted. Table 5–2 shows the standard deviation as well as the mean, maximum, and minimum amount of time taken by agent v_1 to perform a local computation for a given multi-agent network under normal operating conditions. A $500ms$ delay experienced by at least one agent in the $2, 4$ or even 8 agent networks is a relatively plausible occurrence. In larger multi-agent networks, such as the 32 or 64 multi-agent networks, a $500ms$ delay is relatively extreme since there could be more than 3500 events (updates by individual agents) in the time it takes the $500ms$ artificially delayed agent to perform just a single update (cf. Table 5–2). The fact that the asynchronous algorithm is still able to converge in this scenario (in one quarter of the time taken by the state-of-the-art methods) is a testament to its robustness.

Fixed Workload per Agent

In each n-agent multi-agent network, each agent randomly samples 290000 data samples from the dataset to construct its local loss function, akin to a stochastic version of the *overlap regime* studied in depth in the seminal works of Bertsekas and Tsitsiklis [5, 78].

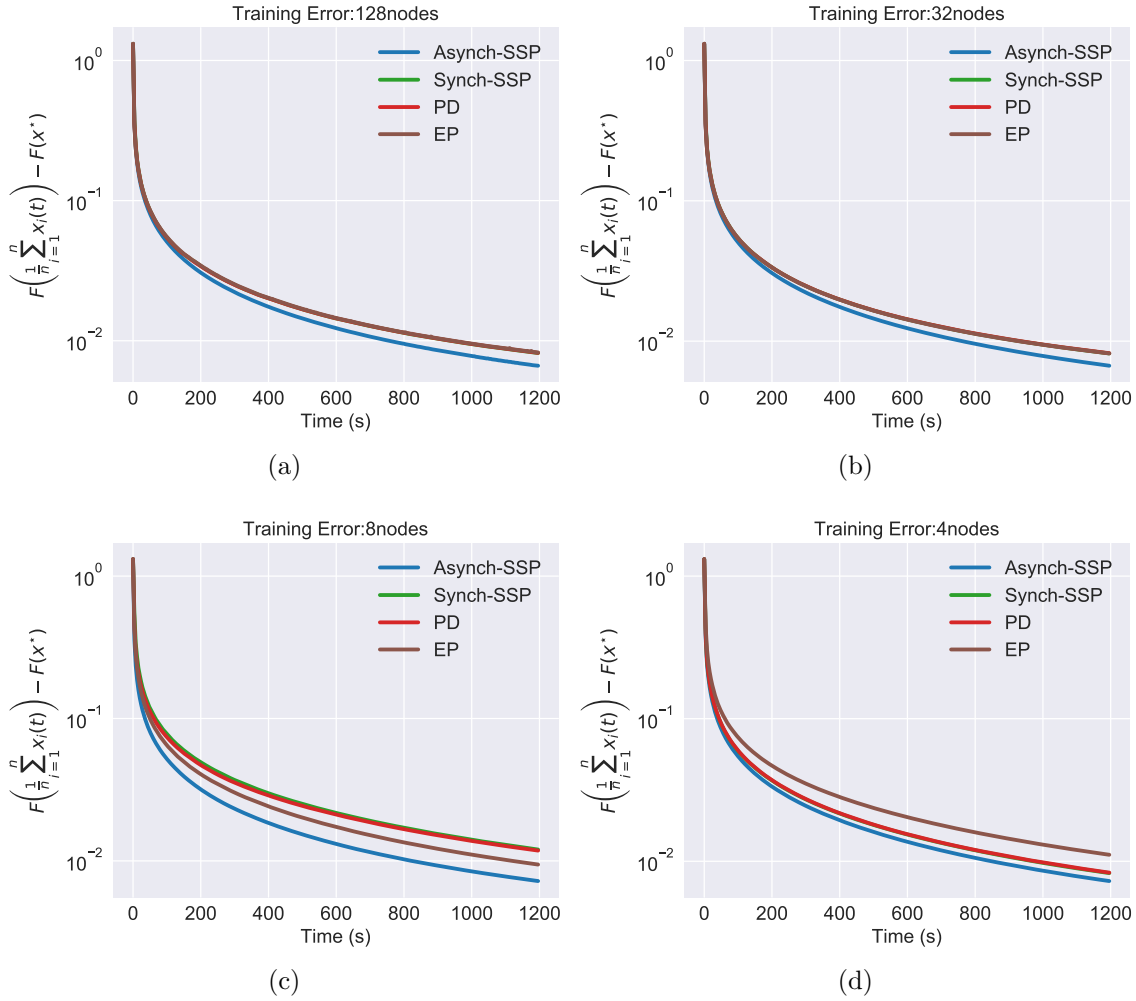


Figure 5–5: Scaling the network size while holding the computational load at each agent fixed. Multinomial logistic regression training error on the covertype dataset, where each agent randomly samples 290000 training instances from the dataset to construct its local loss function. In all cases, the asynchronous algorithm achieves faster convergence than the state-of-the-art methods.

Discussion. Figure 5–5 shows how the relative performance of the algorithms scale with a fixed workload per agent. That is, by keeping the number of data samples per agent fixed and increasing the size of the multi-agent network, we observe that

the asynchronous algorithm consistently achieves faster convergence than the state-of-the-art methods. Most notably, as the computational load per agent increases, the significance of any operating variance decreases (cf. Table 5–2), and thus the effective level of asynchrony in the multi-agent network decreases, thereby leading the asynchronous algorithm to perform reliably with less fluctuations in its optimization trajectory. As a peripheral observation, it is interesting to note that the time taken by the asynchronous algorithm to achieve the target residual error of 0.01 in this *overlap regime* formulation is nearly identical for all reported multi-agent networks (this does not seem to hold up for the synchronous algorithms); one of the main motivations for constructing such an *overlap regime* in practice is to provide increased robustness to individual node failure [6, 5, 78].

CHAPTER 6
Summary and Extensions to Future Work

The burgeoning of distributed optimization is attributable to the myriad applications in which in which such problems arise. Touching domains from finance to the engineering, biological and social sciences, the need to develop distributed optimization methods is rooted in practical applications involving the processing of data that is naturally distributed, private, or simply too large to store on a single machine. More generally, there has always been a need for the solution of very large computational problems, whether data-based or not, and distributed optimization methods provide a practical way to tackle these problems. However, despite the practice-motivated roots of distributed optimization, the crux of the matter is that there remains a nontrivial gap between theory and practice. Relative to the amount of the theoretical development in the literature, there has been less work investigating practical implementations of these algorithms, and furthermore, many of the theoretical analyses makes assumptions that are difficult or undesirable to satisfy in practice (*e.g.*, synchronous, push-pull, doubly-stochastic) [74].

Based on this observation, we extend the baseline Subgradient-Push optimization method to an asynchronous implementation, and show that it out-performs the state-of-the-art (synchronous) methods in practice in terms of optimization time, scalability with the network size, and robustness to failing or stalling agents. We then proceed to develop a convergence theory for Asynchronous Subgradient Push

by developing a general modelling framework for analyzing asynchronous algorithms, and applying it to the analysis of Asynchronous Subgradient Push. Along the way, we also develop some peripheral results concerning the convergence, and convergence rate, of an asynchronous version of the Push-Sum consensus protocol that are interesting in their own respect. We also implement and open-source implementations of the state-of-the-art first-order methods compared in this work using the MPI standard for message-passing in clusters [18, 17, 28], and make them available to the community.[1]

It should be noted that even though our analyses are presented for static communication graphs, they actually hold trivially for *time-varying* communication graphs so long as the graph sequence is "slowly time-varying" and is B-strongly connected for some finite B. Precisely, the reference graph should remain static for at least $\overline{\tau}^{\text{proc}} - 1$ consecutive iterations before changing topology, and there should exist a finite $\widetilde{B}$ such that the union of $(\overline{\tau}^{\text{proc}})\widetilde{B}$ consecutive graphs is strongly connected. If the time-index processing delay, $\overline{\tau}^{\text{proc}}$, is reduced to 1 (synchronous operation), then the proof holds for *all* time-varying graphs that are $\widetilde{B}$-strongly connected for arbitrary $\widetilde{B}$.

In general, motivated by empirical observations [2], this thesis introduces the first of, what will hopefully be, many singly-stochastic asynchronous multi-agent optimization methods available in the literature (cf. Section 1.4). In this work we extended synchronous Subgradient-Push to an asynchronous implementation, but

[1] Open Source: https://github.com/MidoAssran/maopy

Subgradient-Push is simply a multi-agent analog of gradient descent, and it would be interesting to explore the possibility of extending other algorithms to asynchronous operation using singly-stochastic consensus matrices. For example, it would be interesting to explore methods that use an extrapolation between iterates to accelerate convergence [58, 88], or quasi-newton methods that approximate the Hessian using only first-order information [51], or Lagrangian-dual methods that formulate the consensus constrained optimization problems using the Lagrangian, or Augmented Lagrangian, and simultaneously solve for both primal and dual variables [71]. Furthermore, it would be interesting to establish convergence rates for asynchronous versions of these algorithms. Another interesting direction of future work would be to develop some theory for multi-agent optimization algorithms in the non-convex case, given that most of the theoretical development to date has focused on convex optimization. Non-convexity arises quite often in practice, especially in deep-learning applications, which appear to lend themselves quite nicely to multi-agent optimization methods given the scale of computing resources involved.

References

[1] Tansu Alpcan and Christian Bauckhage. A distributed machine learning framework. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 2546–2551. IEEE, 2009.

[2] Mahmoud Assran and Michael G Rabbat. An empirical comparison of multi-agent optimization algorithms. In *IEEE GlobalSIP Symposium on Distributed Optimization and Resource Management over Networks*. IEEE, 2017.

[3] Arda Aytekin. *Asynchronous Algorithms for Large-Scale Optimization: Analysis and Implementation*. PhD thesis, KTH Royal Institute of Technology, 2017.

[4] Florence Bénézit, Vincent Blondel, Patrick Thiran, John Tsitsiklis, and Martin Vetterli. Weighted gossip: Distributed averaging using non-doubly stochastic matrices. In *Information theory proceedings (isit), 2010 ieee international symposium on*, pages 1753–1757. IEEE, 2010.

[5] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

[6] Dimitri P Bertsekas and John N Tsitsiklis. Some aspects of parallel and distributed iterative algorithms—a survey. *Automatica*, 27(1):3–21, 1991.

[7] Pascal Bianchi, Walid Hachem, and Franck Iutzeler. A coordinate descent primal-dual algorithm and application to distributed asynchronous optimization. *IEEE Transactions on Automatic Control*, 61(10):2947–2957, 2016.

[8] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2508–2530, 2006.

[9] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction

method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[10] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[11] Loris Cannelli, Francisco Facchinei, Vyacheslav Kungurtsev, and Gesualdo Scutari. Asynchronous parallel algorithms for nonconvex big-data optimization. part ii: Complexity and numerical results. *arXiv preprint arXiv:1701.04900*, 2017.

[12] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438, 2013.

[13] Valerio Cappellini, Hans-Jürgen Sommers, Wojciech Bruzda, and Karol Życzkowski. Random bistochastic matrices. *Journal of Physics A: Mathematical and Theoretical*, 42(36):365209, 2009.

[14] Volkan Cevher, Stephen Becker, and Mark Schmidt. Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics. *IEEE Signal Processing Magazine*, 31(5):32–43, 2014.

[15] Themistoklis Charalambous and Christoforos N Hadjicostis. Average consensus in the presence of dynamically changing directed topologies and time delays. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 709–714. IEEE, 2014.

[16] Themistoklis Charalambous, Ye Yuan, Tao Yang, Wei Pan, Christoforos N Hadjicostis, and Mikael Johansson. Distributed finite-time average consensus in digraphs in the presence of time delays. *IEEE Transactions on Control of Network Systems*, 2(4):370–381, 2015.

[17] Lisandro Dalcin. mpi4py, 2007.

[18] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. Mpi for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

[19] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[20] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[21] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[22] Paolo Di Lorenzo, Sergio Barbarossa, and Ali H Sayed. Decentralized resource assignment in cognitive networks based on swarming mechanisms over random graphs. *IEEE Transactions on Signal Processing*, 60(7):3755–3769, 2012.

[23] Alexandros G. Dimakis, Soummya Kar, José M. F. Moura, Michael G. Rabbat, and Anna Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98:1847–1864, 2010.

[24] Alexandros G Dimakis, Soummya Kar, José MF Moura, Michael G Rabbat, and Anna Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864, 2010.

[25] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2012.

[26] Mark Eisen, Aryan Mokhtari, and Alejandro Ribeiro. Decentralized quasi-newton methods. *IEEE Transactions on Signal Processing*, 65(10):2613–2628, 2017.

[27] Fabio Fagnani and Sandro Zampieri. Randomized consensus algorithms over large scale networks. *IEEE Journal on Selected Areas in Communications*, 26(4), 2008.

[28] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[29] Christoforos N Hadjicostis and Themistoklis Charalambous. Average consensus in the presence of delays and dynamically changing directed graph topologies. *arXiv preprint arXiv:1210.4778*, 2012.

[30] Christoforos N Hadjicostis and Themistoklis Charalambous. Average consensus in the presence of delays in directed graph topologies. *IEEE Transactions on Automatic Control*, 59(3):763–768, 2014.

[31] John Hajnal and MS Bartlett. Weak ergodicity in non-homogeneous markov chains. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 54, pages 233–246. Cambridge University Press, 1958.

[32] Matthew T Hale, Angelia Nedich, and Magnus Egerstedt. Asynchronous multi-agent primal-dual optimization. *IEEE Transactions on Automatic Control*, 2017.

[33] Robert Hannah and Wotao Yin. More iterations per second, same quality–why asynchronous algorithms may drastically outperform traditional ones. *arXiv preprint arXiv:1708.05136*, 2017.

[34] Jarvis D. Haupt, W. U. Bajwa, Michael G. Rabbat, and Ryan K. Nowak. Compressed sensing for networked data. *IEEE Signal Processing Magazine*, 25:92–101, 2008.

[35] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex analysis and minimization algorithms I: Fundamentals*, volume 305. Springer science & business media, 2013.

[36] Mingyi Hong and Tsung-Hui Chang. Stochastic proximal gradient consensus over random networks. *IEEE Transactions on Signal Processing*, 65(11):2933–2948, 2017.

[37] Franck Iutzeler, Pascal Bianchi, Philippe Ciblat, and Walid Hachem. Asynchronous distributed optimization using a randomized alternating direction method of multipliers. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 3671–3676. IEEE, 2013.

[38] Soummya Kar and José MF Moura. Sensor networks with random links: Topology design for distributed consensus. *IEEE Transactions on Signal Processing*, 56(7):3315–3326, 2008.

[39] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482–491. IEEE, 2003.

[40] Sandeep Kumar, Rahul Jain, and Ketan Rajawat. Asynchronous optimization over heterogeneous networks via consensus admm. *IEEE Transactions on Signal and Information Processing over Networks*, 3(1):114–129, 2017.

[41] Rafael Laufer, Henri Dubois-Ferriere, and Leonard Kleinrock. Multirate anypath routing in wireless mesh networks. In *INFOCOM 2009, IEEE*, pages 37–45. IEEE, 2009.

[42] Na Li, Lijun Chen, and Steven H Low. Optimal demand response based on utility maximization in power networks. In *Power and Energy Society General Meeting, 2011 IEEE*, pages 1–8. IEEE, 2011.

[43] Shu Li and Tamer Basar. Asymptotic agreement and convergence of asynchronous stochastic algorithms. *IEEE Transactions on Automatic Control*, 32(7):612–618, 1987.

[44] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[45] M. Lichman. UCI machine learning repository, 2013.

[46] Pedro U Lima and Luis M Custodio. Multi-robot systems. In *Innovations in robot mobility and control*, pages 1–64. Springer, 2005.

[47] Ji Liu and Stephen J Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 25(1):351–376, 2015.

[48] Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 16(285-322):1–5, 2015.

[49] Qingguo Lü and Huaqing Li. Geometrical convergence rate for distributed optimization with time-varying directed graphs and uncoordinated step-sizes. *arXiv preprint arXiv:1611.00990*, 2016.

[50] Fatemeh Mansoori and Ermin Wei. Superlinearly convergent asynchronous distributed network newton method. *arXiv preprint arXiv:1705.03952*, 2017.

[51] Aryan Mokhtari, Qing Ling, and Alejandro Ribeiro. Network newton distributed optimization methods. *IEEE Transactions on Signal Processing*, 65(1):146–161, 2016.

[52] Angelia Nedic. On the rate of convergence of distributed subgradient methods for multi-agent optimization. In *Decision and Control, 2007 46th IEEE Conference on*, pages 4711–4716. IEEE, 2007.

[53] Angelia Nedic. Asynchronous broadcast-based convex optimization over a network. *IEEE Transactions on Automatic Control*, 56(6):1337–1351, 2011.

[54] Angelia Nedić. Distributed optimization. *Encyclopedia of Systems and Control*, pages 308–317, 2015.

[55] Angelia Nedić and Alex Olshevsky. Stochastic gradient-push for strongly convex functions on time-varying directed graphs. *IEEE Transactions on Automatic Control*, 61(12):3936–3947, 2016.

[56] Angelia Nedic and Asuman Ozdaglar. 10 cooperative distributed multi-agent. *Convex Optimization in Signal Processing and Communications*, 340, 2010.

[57] A. Nedich and A. Olshevsky. Distributed optimization over time-varying directed graphs. *IEEE Transactions on Automatic Control*, 60(3):601–615, 2015.

[58] Angelia Nedich, Alex Olshevsky, and Wei Shi. Achieving geometric convergence for distributed optimization over time-varying graphs. *arXiv preprint arXiv:1607.03218*, 2016.

[59] Angelia Nedich, Alex Olshevsky, and Wei Shi. A geometrically convergent method for distributed optimization over time-varying graphs. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 1023–1029. IEEE, 2016.

[60] Giovanni Neglia, Giuseppe Reina, and Sara Alouf. Distributed gradient optimization for epidemic routing: A preliminary evaluation. In *Wireless Days (WD), 2009 2nd IFIP*, pages 1–6. IEEE, 2009.

[61] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[62] Alex Olshevsky. Efficient information aggregation strategies for distributed control and signal processing. *arXiv preprint arXiv:1009.6036*, 2010.

[63] Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin. On the convergence of asynchronous parallel iteration with arbitrary delays. *arXiv preprint arXiv:1612.04425*, 2016.

[64] Joel B Predd, Sanjeev R Kulkarni, and H Vincent Poor. A collaborative training algorithm for distributed learning. *IEEE Transactions on Information Theory*, 55(4):1856–1871, 2009.

[65] Michael Rabbat and Robert Nowak. Distributed optimization in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 20–27. ACM, 2004.

[66] Michael G Rabbat and Konstantinos I Tsianos. Asynchronous decentralized optimization in heterogeneous systems. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 1125–1130. IEEE, 2014.

[67] S Sundhar Ram, Angelia Nedić, and Venugopal V Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147(3):516–545, 2010.

[68] Sundhar Srinivasan Ram, Venugopal V Veeravalli, and Angelia Nedic. Distributed non-autonomous power control through distributed convex optimization. In *INFOCOM 2009, IEEE*, pages 3001–3005. IEEE, 2009.

[69] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[70] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):994–966, 2015.

[71] Wei Shi, Qing Ling, Kun Yuan, Gang Wu, and Wotao Yin. On the linear convergence of the admm in decentralized consensus optimization. *IEEE Transactions on Signal Processing*, 62(7):1750–1761, 2014.

[72] Konstantinos Tsianos. *The role of the network in distributed optimization algorithms: Convergence rates, scalability, communication/computation tradeoffs and communication delays*. PhD thesis, McGill University Libraries, 2013.

[73] Konstantinos Tsianos, Sean Lawlor, and Michael G Rabbat. Communication/computation tradeoffs in consensus-based distributed optimization. In *Advances in neural information processing systems*, pages 1943–1951, 2012.

[74] Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat. Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 1543–1550. IEEE, 2012.

[75] Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat. Push-sum distributed dual averaging for convex optimization. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 5453–5458, 2012.

[76] Konstantinos I Tsianos and Michael G Rabbat. The impact of communication delays on distributed consensus algorithms. *arXiv preprint arXiv:1207.5839*, 2012.

[77] Konstantinos I Tsianos and Michael G Rabbat. Efficient distributed online prediction and stochastic optimization with approximate distributed averaging. *IEEE Transactions on Signal and Information Processing over Networks*, 2(4):489–506, 2016.

[78] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[79] Ermin Wei and Asuman Ozdaglar. On the o (1= k) convergence of asynchronous distributed alternating direction method of multipliers. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 551–554. IEEE, 2013.

[80] Jacob Wolfowitz. Products of indecomposable, aperiodic, stochastic matrices. *Proceedings of the American Mathematical Society*, 14(5):733–737, 1963.

[81] Qihui Wu, Guoru Ding, Yuhua Xu, Shuo Feng, Zhiyong Du, Jinlong Wang, and Keping Long. Cognitive internet of things: a new paradigm beyond connection. *IEEE Internet of Things Journal*, 1(2):129–143, 2014.

[82] Tianyu Wu, Kun Yuan, Qing Ling, Wotao Yin, and Ali H Sayed. Decentralized consensus optimization with asynchrony and delay. 2016.

[83] Tianyu Wu, Kun Yuan, Qing Ling, Wotao Yin, and Ali H Sayed. Decentralized consensus optimization with asynchrony and delays. In *Signals, Systems and Computers, 2016 50th Asilomar Conference on*, pages 992–996. IEEE, 2016.

[84] Chenguang Xi. *Distributed Optimization Algorithms in Large-Scale Directed Networks*. PhD thesis, Tufts University, 2017.

[85] Chenguang Xi and Usman A Khan. Dextra: A fast algorithm for optimization over directed graphs. *IEEE Transactions on Automatic Control*, 2017.

[86] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(Oct):2543–2596, 2010.

[87] Yuhua Xu, Jinlong Wang, Qihui Wu, Alagan Anpalagan, and Yu-Dong Yao. Opportunistic spectrum access in cognitive radio networks: Global optimization using local interaction games. *IEEE Journal of Selected Topics in Signal Processing*, 6(2):180–194, 2012.

[88] J. Zeng and W. Yin. Extra push for convex smooth decentralized optimization over directed networks. *UCLA CAM Report*, 15-61, 2015. http://arxiv.org/abs/1511.02942.

[89] Jinshan Zeng and Tao He. A fast algorithm for distributed optimization over directed networks. In *IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 45–49. IEEE, 2016.

[90] Guoqiang Zhang and Richard Heusdens. Bi-alternating direction method of multipliers over graphs. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 3571–3575. IEEE, 2015.