2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

ASY-SONATA: Achieving Linear Convergence in Distributed Asynchronous Multiagent Optimization

Ye Tian, Ying Sun, and Gesualdo Scutari

Abstract— **This papers studies multi-agent (convex and *nonconvex*) optimization over static digraphs. We propose a general distributed *asynchronous* algorithmic framework whereby i) agents can update their local variables as well as communicate with their neighbors at any time, without any form of coordination; and ii) they can perform their local computations using (possibly) delayed, out-of-sync information from their neighbors. Delays need not be known to the agents or obey any specific profile, and can also be time-varying (but bounded). The algorithm builds on a tracking mechanism that is robust against asynchrony (in the above sense), whose goal is to estimate locally the sum of agents' gradients. When applied to strongly convex functions, we prove that it converges at an R-linear (geometric) rate as long as the step-size is sufficiently small. A sublinear convergence rate is proved, when nonconvex problems and/or diminishing, *uncoordinated* step-sizes are employed. To the best of our knowledge, this is the first distributed algorithm with provable geometric convergence rate in such a general asynchonous setting.**

I. Introduction

In this paper, we study (convex and nonconvex) distributed optimization over a network of agents, modeled as a directed, fixed, graph. Agents aim at cooperatively solving the following optimization problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} F(\mathbf{x}) \triangleq \sum_{i=1}^{I} f_i(\mathbf{x}) \tag{P}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ is the cost function of agent i, assumed to be smooth (nonconvex) and known only to agent i. In this setting, optimization has to be performed in a distributed, collaborative manner: agents can only receive/send information from/to its immediate neighbors. Instances of (P) that require distributed computing have found a wide range of applications in different areas, including network information processing, resource allocation in communication networks, swarm robotic, and machine learning, just to name a few.

Many of the aforementioned applications give rise to extremely large-scale problems and networks, which naturally call for *asynchronous*, *parallel* solution methods. In fact, asynchronous methods reduce the idle times of workers, mitigate communication and/or memory-access congestion, save power (as agents need not perform computations and communications at every iteration), and make algorithms more fault-tolerant. In this paper, we consider the following very general, abstract, asynchronous model [1]:

This work has been supported by the USA National Science Foundation under Grants CIF 1632599 and CIF 1719205; and in part by the Office of Naval Research under Grant N00014-16-1-2244, and the Army Research Office under Grant W911NF1810238.

The authors are with the School of Industrial Engineering, Purdue University, West-Lafayette, IN, USA; Emails: {tian110,sun578,gscutari}@purdue.edu.

(i) Agents can perform their local computations as well as communicate (possibly in parallel) with their immediate neighbors at any time, without any form of coordination or centralized scheduling; and
(ii) when solving their local subproblems, agents can use outdated information from their neighbors.

In (ii) no constraint is imposed on the delay profiles: delays can be arbitrary (but bounded), time-varying, and (possibly) dependent on the specific activation rules adopted to wakeup the agents in the network. This model captures in a unified fashion several forms of asynchrony: some agents execute more iterations than others; some agents communicate more frequently than others; and inter-agent communications can be unreliable and/or subject to unpredictable, unknown, (possibly) time-varying delays.

Several forms of asynchrony have been studied in the literature–see Sec. I-A for an overview of related works. However, we are not aware of any distributed algorithm that is compliant to the asynchrony model (i)-(ii) and distributed (nonconvex) setting above. Furthermore, when considering the special case of strongly convex function F, it is not clear how to design a (first-order) *distributed asynchronous* algorithm (as specified above) that achieves *linear convergent* rate over *digraphs*. This paper provides a positive answer to these questions–see Sec. I-B and Table 1 for a summary of our contributions.

A. Literature Review

Since the seminal work [9], asynchronous parallelism has been applied to several *centralized* optimization algorithms, including block coordinate descent (e.g., [9]–[11]) and stochastic gradient (e.g., [12]–[14]) methods. However, these schemes are not applicable to the networked, distributed setup considered in this paper, because they would require the knowledge of the entire function F from each agent. Some of these schemes were extended to hierarchical networks (e.g., master-slave architectures and star networks), see [15], [16], and references therein. However, they remain *centralized*, due to the use of a master (or cluster-head) node.

Distributed methods exploring (some form of) asynchrony over networks with no centralized node have been studied in [2]–[8], [17]–[27]. We group next these works based upon the features (i)-(ii) above.

(a) Random activations and no delays [17]–[21]: These schemes considered distributed *convex* unconstrained optimization over *undirected* graphs. While substantially different in the form of the updates performed by the agents– [17], [19], [21] are instances of primal-dual (proximal-based) algo-

Algorithm	Nonconvex Cost Function	No Idle Time	Arbitrary Delays	Parallel	Step Sizes		Digraph	Global Convergence to Exact Solutions	Rate Analysis	
					Fixed	Uncoordinated Diminishing			Linear Rate for Strongly Convex	Nonconvex
Asyn. Broadcast [2]				✓	✓	✓		In expectation (w. diminishing step)		
Asyn. Diffusion [3]					✓					
Asyn. ADMM [4]	✓				✓			Deterministic		
Dual Ascent in [5]		✓	Restricted	Restricted	✓					
ra-NRC [6]					✓		✓			
ARock [7]		✓	Restricted		✓			Almost surely	In expectation	
ASY-PrimalDual [8]		✓	Restricted		✓			Almost surely		
ASY-SONATA	✓	✓	✓	✓	✓	✓	✓	Deterministic	Deterministic	Deterministic

TABLE 1: Comparison with state-of-art distributed algorithms considering asynchrony and (some forms of) delays. The proposed algorithmic framework–ASY-SONATA–enjoys all the desirable features listed in the table.

rithms, [20] is an ADMM-type algorithm, while [18] is based on the distributed gradient tracking mechanism introduced in [28]–[30]–all these algorithms are asynchronous in the sense of feature (i) [but not (ii)]: at each iteration, a subset of agents [17], [19], [21] (or edge-connected agents [18], [20]), chosen at random, is activated, performing then their updates and communications with their immediate neighbors; between two activations, agents are assumed to be in *idle* mode (i.e., able to *continuously* receive information from their neighbors). However, *no form of delays* is allowed: every agent must perform its local computations/updates using the *most updated* information from its neighbors. This means that all the actions performed by the agent(s) in an activation must be completed before a new activation (agent) can take place (wakes-up), which calls for some coordination among the agents. Finally, no convergence rate was provided for the aforementioned schemes but [18], [20].

(b) Synchronous activations and delays [22]–[27]: These schemes considered distributed constrained *convex* optimization over *undirected* graphs. They study the impact of delayed gradient information [22], [23] or communication delays (fixed [24], uniform [23], [27] or time-varying [25], [26]) on the convergence rate of distributed gradient (proximal [22], [23] or projection-based [26], [27]) algorithms or dual-averaging distributed-based schemes [24], [25]. While these schemes are all synchronous [thus lacking of feature (i)], they can tolerate *communication delays* [an instantiation of feature (ii)], still converging at a *sublinear rate* to an optimal solution of the problem. Delay profiles cannot be arbitrary, but such that no losses occur–every agent's message will eventually reach its destination within a finite time.

(c) Random/cyclic activations and some form of delays [2]–[8]: The class of optimization problems along with the key features of the algorithms proposed in these papers are summarized in Table 1 and briefly discussed next. The majority of these works studied distributed (strongly) *convex* optimization over *undirected* graphs, with [3] assuming that all the functions f_i have the same minimizer, [4] considering also nonconvex objectives, and [6] being implementable also over digraphs. The algorithms in [2], [3] are gradient-based schemes; [4] is a decentralized instance of ADMM; [7] applies an asynchronous parallel ADMM scheme to distributed optimization; and [8] builds on a primal-dual method. The schemes in [5], [6] instead build on (approximate) second-order information. All these algorithms are asynchronous in the sense of feature (i): [2]–[4], [7], [8] considered random activations of the agents (or edges-connected agents) while [5], [6] studied deterministic, uncoordinated activation rules. As far as feature (ii) is concerned, some form of delays is allowed. More specifically, [2]–[4], [6] can deal with *packet losses*: the information sent by an agent to its neighbors either gets lost or received with *no delay*. They also assume that agents are *always in idle mode* between two activations. Closer to the proposed asynchronous framework are the schemes in [7], [8] wherein a probabilist model is employed to describe the activation of the agents and the aged information used in their updates. The model requires that the random variables triggering the activation of the agents are i.i.d and *independent* of the delay vector used by the agent to performs its update. As observed by the authors themselves, while this assumption makes the convergence analysis possible, in reality, there is a strong dependence of the delays on the activation index; see [11] for a detailed discussion on this issue and several examples. Other consequences of this model are: the schemes [7], [8] are *not parallel*–only one agent per time can perform the update–and a random self-delay is injected in the update of each agent (even if agents have access to their most recent information). Finally, referring to the convergence rate, [7] is the only scheme with provably convergence rate: when F is strongly convex, [7] achieves linear convergence rate *in expectation (only)*. No convergence rate is available in any of the aforementioned papers, when F is nonconvex.

B. Summary of Contributions

This paper proposes a general distributed, asynchronous algorithmic framework for convex and *nonconvex* instances of Problem (P), over *directed* graphs. The algorithm leverages a perturbed "sum-push" mechanism that is robust against asynchrony, whose goal is to track locally the average of agents' gradients; this sum-push scheme builds on the idea first introduced in [31] and further developed in [6], [32], with some important differences (cf. Remark 3.3, Sec. III-A). To the best of our knowledge, the proposed framework is the first scheme combining the following attractive features (see also Table 1): (a) it is *parallel and asynchronous [in the sense (i) and (ii)]*–multiple agents can be activated at the same time (with no coordination) and/or outdated information can be used in the agents' updates; our asynchronous setting (i) and (ii) is less restrictive than the one in [7], [8]; furthermore, in contrast with [7], our scheme avoids solving possibly complicated subproblems; (b) it is applicable to *nonconvex* problems; (c) it is implementable over *digraph*;

(d) it employs either a constant step-size or *uncoordinated* diminishing ones; (e) it *convergences at an R-linear rate* (resp. sublinear) when F is strongly convex (resp. nonconvex) and a constant (resp. diminishing, uncoordinated) step-size(s) is employed; and (d) it is "protocol-free", meaning that agents need not obey any specific communication protocols or asynchronous modus operandi (as long as delays are bounded and agents update/communicate uniformly infinitely often), which otherwise would impose some form of coordination.

II. Problem Setup and Preliminaries

A. Problem Setup

We study Problem (P) under the following assumptions.

Assumption 2.1 (On the optimization problem):

(i) Each $f_i : \mathbb{R}^n \to \mathbb{R}$ is proper, closed and L_i-Lipschitz differentiable;

(ii) F is bounded from below. □

Note that f_i need not be convex. We also make the blanket assumption that each agent i knows only its own f_i, but not $\sum_{j\neq i} f_j$. To state linear convergence, we will use the following extra condition on the objective function.

Assumption 2.2 (Strongly convexity): Assumption 2.1(i) holds and, in addition, F is τ-strongly convex. □

On the communication network: The communication network of the agents is modeled as a fixed, directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, I\}$ is the set of nodes (agents), and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges (communication links). If $(i, j) \in \mathcal{E}$, it means that agent i can send information to agent j. We assume that the digraph does not have self-loops. We denote by $\mathcal{N}_i^{\text{in}}$ the set of *in-neighbors* of node i, i.e., $\mathcal{N}_i^{\text{in}} \triangleq \{j \in \mathcal{V} \mid (j, i) \in \mathcal{E}\}$ while $\mathcal{N}_i^{\text{out}} \triangleq \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$ is the set of *out-neighbors* of agent i. We make the following standard assumption on the graph connectivity.

Assumption 2.3: The graph $\mathcal{G}$ is strongly connected. □

B. Preliminaries: The SONATA algorithm [33], [34]

The asynchronous, distributed framework we are going to introduce builds on the synchronous SONATA algorithm, proposed in [33], [34] to solve (nonconvex) multi-agent optimization problems over time-varying digraphs. This is motivated by the fact that SONATA has the unique property of being provably applicable to both convex and nonconvex problems, and it achieves liner convergence when applied to strongly convex objectives F. We thus begin reviewing a special instance of SONATA, tailored to (P). In Sec. IV, we will make such a scheme asynchronous.

Every agent controls and iteratively updates the tuple $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i, \phi_i)$: $\mathbf{x}_i$ is agent i's copy of the shared variables $\mathbf{x}$ in (P); $\mathbf{y}_i$ acts as a local proxy of the sum-gradient ∇F; and $\mathbf{z}_i$ and ϕ_i are auxiliary variables instrumental to deal with communications over digraphs (their goal will be clear shortly). Let $\mathbf{x}_i^k, \mathbf{z}_i^k$, ϕ_i^k, and $\mathbf{y}_i^k$ denote the value of the aforementioned variables at iteration $k \in \mathbb{N}_+$. The update of each agent i reads:

$$\mathbf{x}_i^{k+1} = \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} w_{ij}\left(\mathbf{x}_j^k - \alpha^k\,\mathbf{y}_j^k\right), \tag{1}$$

$$\mathbf{z}_i^{k+1} = \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} a_{ij}\left(\mathbf{z}_j^k + \nabla f_j(\mathbf{x}_j^{k+1}) - \nabla f_j(\mathbf{x}_j^k)\right), \tag{2}$$

$$\phi_i^{k+1} = \sum_{j\in\mathcal{N}_i^{\text{in}}\cup\{i\}} a_{ij}\phi_j^k, \tag{3}$$

$$\mathbf{y}_i^{k+1} = \mathbf{z}_i^{k+1}/\phi_i^{k+1}, \tag{4}$$

with $\mathbf{z}_i^0 = \mathbf{y}_i^0 = \nabla f_i(\mathbf{x}_i^0)$ and $\phi_i^0 = 1$, for all $i \in \mathcal{V}$. In (1), $\mathbf{y}_i^k$ is a local estimate of the average-gradient $(1/I)\sum_{i=1}^I \nabla f_i(\mathbf{x}_i^k)$. Therefore, every agent, first moves along the estimated gradient direction, generating $\mathbf{x}_i^k - \alpha^k\,\mathbf{y}_i^k$ (α^k is the step-size); and then performs a consensus step to force asymptotic agreement among the local variables $\mathbf{x}_i$. Steps (2)-(4) represent a perturbed-push-sum update, aiming at tracking the gradient $(1/I)\,\nabla F$ [29], [30], [34]. The weight-matrices $\mathbf{W} \triangleq (w_{ij})_{i,j=1}^I$ and $\mathbf{A} \triangleq (a_{ij})_{i,j=1}^I$ satisfy the following standard assumptions.

Assumption 2.4: (On the weight-matrices) The weigh-matrices $\mathbf{W} \triangleq (w_{ij})_{i,j=1}^I$ and $\mathbf{A} \triangleq (a_{ij})_{i,j=1}^I$ satisfy (we will write $\mathbf{M} \triangleq (m_{ij})_{i,j=1}^I$ to denote either $\mathbf{A}$ or $\mathbf{W}$):

(i) $\exists\, \bar{m} > 0$ such that $m_{ii} \geq \bar{m}$, for all $i \in \mathcal{V}$; and $m_{ij} \geq \bar{m}$, for all $(j, i) \in \mathcal{E}$; $m_{ij} = 0$, otherwise;

(ii) $\mathbf{W}$ is row-stochastic, that is, $\mathbf{W}\,\mathbf{1} = \mathbf{1}$;

(iii) $\mathbf{A}$ is column-stochastic, that is, $\mathbf{A}^T\,\mathbf{1} = \mathbf{1}$; □

In [35], the authors proved that a special instance of SONATA, when applied to (P) with strongly convex F, converges at an R-linear rate. This result was further extended to constraints, nonsmooth, distributed optimization in [36].

How difficult is making SONATA asynchronous? A natural question is whether one can build on SONATA to design an asynchronous scheme that still enjoys linear convergence rate. Note that naive modifications of the updates (1)-(4) to make them asynchronous–such as using uncoordinated activations and/or delayed information–may not work, because they would affect some key properties of the updates that guarantee convergence. For instance, the tracking (2)-(4) calls for the invariance of the averages, i.e., $\sum_{i=1}^I \mathbf{z}_i^k = \sum_{i=1}^I \nabla f_i(\mathbf{x}^k)$, for all $k \in \mathbb{N}_+$. It is not difficult to check that any perturbation injected in (2)-e.g., in the form of delays or packed losses–puts in jeopardy the aforementioned property.

To cope with the above challenges, a first step is robustifying the gradient tracking scheme. In Sec. III, we introduce P-ASY-SUM-PUSH–an asynchronous, perturbed, instance of the push-sum algorithm [37] (written in the "sum-push" form), which also serves as a novel gradient tracking mechanism that is robust against asynchronous activations and delays. Building on this result, in Sec. IV, we present the proposed distributed asynchronous optimization framework, termed ASY-SONATA.

III. Perturbed Asynchronous Sum-Push

In this section, we build P-ASY-SUM-PUSH. For the sake of clarity, we first introduce the *synchronous* push-sum in the

"sum-push" form; we then break the synchronism.

Preliminaries: the sum-push algorithm. Consider the average consensus problem in the multi-agent setting introduced in Sec. II-A. This problem can be solved using the push-sum algorithm [37]. In view of our asynchronous implementation, it is convenient to rewrite the push-sum algorithm breaking the "push" and "sum" steps in two *separate* actions and switch their order. While there is no advantage in doing that in a synchronous setting, this will simplify the presentation of its asynchronous counterpart as well as lead to a more flexible asynchronous implementation.

The sum-push reads: given $\mathbf{z}_i^k$, ϕ_i^k, $\boldsymbol{\rho}_{ij}^k$, and σ_{ij}^k, at iteration $k \in \mathbb{N}_+$, each agent $i \in \mathcal{V}$ performs

$$\text{Sum:} \quad \begin{cases} \mathbf{z}_i^{k+\frac{1}{2}} = \mathbf{z}_i^k + \sum_{j \in \mathcal{N}_i^{\text{in}}} \boldsymbol{\rho}_{ij}^k, \\ \phi_i^{k+\frac{1}{2}} = \phi_i^k + \sum_{j \in \mathcal{N}_i^{\text{in}}} \sigma_{ij}^k; \end{cases} \tag{5}$$

$$\text{Push:} \quad \begin{cases} \mathbf{z}_i^{k+1} = a_{ii}\, \mathbf{z}_i^{k+\frac{1}{2}}, \\ \phi_i^{k+1} = a_{ii}\, \phi_i^{k+\frac{1}{2}}, \\ \boldsymbol{\rho}_{ji}^{k+1} = a_{ji}\, \mathbf{z}_i^{k+\frac{1}{2}}, \quad \forall j \in \mathcal{N}_i^{\text{out}}, \\ \sigma_{ji}^{k+1} = a_{ji}\, \phi_i^{k+\frac{1}{2}}, \quad \forall j \in \mathcal{N}_i^{\text{out}}; \end{cases} \tag{6}$$

$$\mathbf{y}_i^{k+1} = \frac{\mathbf{z}_i^{k+1}}{\phi_i^{k+1}}; \tag{7}$$

where $\phi_i^0 = 1$, $\boldsymbol{\rho}_{ij}^k = \mathbf{0}$, and $\sigma_{ij}^0 = 0$, for all $(j,i) \in \mathcal{E}$, and the weight-matrix $\mathbf{A} \triangleq (a_{ij})_{i,j=1}^I$ satisfies Assumption 2.4(i),(iii). In words, at iteration k, every agent i first performs the "sum" step (5) and builds the new mass $\mathbf{z}_i^{k+1/2}$: it sums its current information $\mathbf{z}_i^k$ with the one sent from its in-neighbors–$\boldsymbol{\rho}_{ij}^k$ is the information sent to i by agent $j \in \mathcal{N}_i^{\text{in}}$. Then, the "push" step (6) follows: $\mathbf{z}_i^{k+1/2}$ is "pushed back" (sent) to the out-neighbors $j \in \mathcal{N}_i^{\text{out}}$ and agent i itself; out of the total mass $\mathbf{z}_i^{k+1/2}$, each $j \in \mathcal{N}_i^{\text{out}}$ receives the fraction $\boldsymbol{\rho}_{ji}^{k+1} = a_{ji}\, \mathbf{z}_i^{k+1/2}$, with agent i getting $a_{ii}\, \mathbf{z}_i^{k+1/2}$, which determines the update $\mathbf{z}_i^k \to \mathbf{z}_i^{k+1}$. It is not difficult to check that the overall mass in the system does not change over the time and equals the initial mass:

$$\sum_{i=1}^I \mathbf{z}_i^{k+1} + \sum_{(j,i)\in\mathcal{E}} \boldsymbol{\rho}_{ij}^{k+1} = \sum_{i=1}^I \mathbf{z}_i^k + \sum_{(j,i)\in\mathcal{E}} \boldsymbol{\rho}_{ij}^k = \cdots = \sum_{i=1}^I \mathbf{z}_i^0.$$

The ϕ, σ-variables satisfy a similar property.

Finally, consistently with the push-sum, the y-variables in (7), can be regarded as agent i's estimate of the average. In fact, it is not difficult to check that, if a consensus is achieved on the $\mathbf{y}_i$'s, i.e., $\lim_{k\to\infty} \mathbf{z}_i^{k+1}/\phi_i^{k+1} = \mathbf{c}^\infty$ for all $i \in \mathcal{V}$, then it must be $\mathbf{c}^\infty = (1/I) \cdot \sum_{i=1}^I \mathbf{z}_i^0$.

Next, we break the synchronism in the sum-push scheme.

Asynchronous sum-push: Consider the following asynchronous setting: i) multiple agents compute and communicate independently without coordination; ii) communication latency and uncoodinated computations result in (possibly time-varying) delays. This means that some agents can execute more iterations than others and, in general they no longer use the most recent information from its neighbors; also, some information can get lost. As a consequence, the key property of the synchronous sum-push–the preservation of the overall mass–would not be guaranteed.

We robustify the sum-push building on the idea first introduced in [31] and further developed in [6], [32]: each $\boldsymbol{\rho}_{ji}$ (resp. σ_{ji}) no longer represents the *current* mass-fraction $a_{ji}\, \mathbf{z}_i$ (resp. $a_{ji}\, \phi_i$), meant for node $j \in \mathcal{N}_i^{\text{out}}$, but it is instead the *running-sum* of the mass $a_{ji}\, \mathbf{z}_i$ (resp. $a_{ji}\, \phi_i$) that has been generated for j *up to the current activation* of agent i. In addition, every agent i maintains, for every $j \in \mathcal{N}_i^{\text{in}}$, a local buffer $\tilde{\boldsymbol{\rho}}_{ij}$ (resp. $\tilde{\sigma}_{ij}$) storing the value of $\boldsymbol{\rho}_{ij}$ (resp. σ_{ij}) that it has used in its *last* (past) update. With this construction, we build next one iteration of the asynchronous sum-push algorithm. We discuss the updates of the $\boldsymbol{z}$, $\boldsymbol{\rho}$, and $\tilde{\boldsymbol{\rho}}$-variables only; the one of the ϕ, σ, and $\tilde{\sigma}$ follows the same argument.

Suppose agent i^k wakes up at iteration k. The state of agent i^k is described by the variables $\mathbf{z}_{i^k}$, $\boldsymbol{\rho}_{ji^k}$, and $\tilde{\boldsymbol{\rho}}_{i^k j}$. However, the $\boldsymbol{\rho}$-variables may no longer contain the current information from its in-neighbors. More specifically, agent i^k does not have access to the current vector $\boldsymbol{\rho}_{i^k j}^k$ from $j \in \mathcal{N}_{i^k}^{\text{in}}$, but it will use instead the delayed version $\boldsymbol{\rho}_{i^k j}^{k-d_j^k}$, where $0 \le d_{ij}^k \le D$ is the delay (assumed to be bounded). By definition, the local buffer $\tilde{\boldsymbol{\rho}}_{i^k j}^k$ stores the value of $\boldsymbol{\rho}_{i^k j}$ that agent i^k used in its previous update. If the information in $\boldsymbol{\rho}_{i^k j}^{k-d_j^k}$ is not older than the one in $\tilde{\boldsymbol{\rho}}_{i^k j}^k$, the difference $\boldsymbol{\rho}_{i^k j}^{k-d_j^k} - \tilde{\boldsymbol{\rho}}_{i^k j}^k$ will capture the sum of the $a_{i^k j}\mathbf{z}_j$'s that have been generated by $j \in \mathcal{N}_{i^k}^{\text{in}}$ for i^k up until $k - d_j^k$ and not used by agent i^k yet; otherwise $\boldsymbol{\rho}_{i^k j}^{k-d_j^k}$ will be discarded, as no new information has been acquired. For instance, in a synchronous setting, one would have $\boldsymbol{\rho}_{ij}^k - \tilde{\boldsymbol{\rho}}_{ij}^k = a_{ij}\mathbf{z}_j^k$. This naturally suggests the following modification of the steps (5)-(7) to preserve the total mass of the system at every iteration:

$$\text{Sum:} \quad \mathbf{z}_{i^k}^{k+\frac{1}{2}} = \mathbf{z}_{i^k}^k + \sum_{j \in \mathcal{N}_{i^k}^{\text{in}}} \left(\boldsymbol{\rho}_{i^k j}^{k-d_j^k} - \tilde{\boldsymbol{\rho}}_{i^k j}^k \right), \tag{8}$$

$$\text{Push:} \quad \begin{cases} \mathbf{z}_{i^k}^{k+1} = a_{i^k i^k}\, \mathbf{z}_{i^k}^{k+\frac{1}{2}}, \\ \boldsymbol{\rho}_{ji^k}^{k+1} = \boldsymbol{\rho}_{ji^k}^k + a_{ji^k}\, \mathbf{z}_{i^k}^{k+\frac{1}{2}}, \quad \forall j \in \mathcal{N}_{i^k}^{\text{out}}; \end{cases} \tag{9}$$

$$\text{Mass-buffer:} \quad \tilde{\boldsymbol{\rho}}_{i^k j}^{k+1} = \boldsymbol{\rho}_{i^k j}^{k-d_j^k}, \qquad \forall j \in \mathcal{N}_{i^k}^{\text{in}} \tag{10}$$

while $\mathbf{y}_{i^k}^{k+1} = \mathbf{z}_{i^k}^{k+1}/\phi_{i^k}^{k+1}$ [cf. (7)]; where $\phi_i^0 = 1$, for all $i \in \mathcal{V}$, and $\boldsymbol{\rho}_{ij}^k = \tilde{\boldsymbol{\rho}}_{ij}^k = \mathbf{0}$, for all $k = -D, \ldots, 0$, and $(j,i) \in \mathcal{E}$. Note that, differently from the synchronous case [cf. (6)], in (9), $\boldsymbol{\rho}_{ji}^k$ is now updated *recursively*, to build the running-sum of the mass $a_{ji}\mathbf{z}_i$. After the sum-step, in (10), the buffer is updated to account for the use of new information from $j \in \mathcal{N}_{i^k}^{\text{in}}$.

With the above modifications, the total mass in the systems is preserved at each iteration, as shown next. Consider only

the $\boldsymbol{z}$-variables; similar argument applies to the ϕ-variables. The total mass associated with the z-variables at iteration k is defined as

$$\mathfrak{m}_z^k \triangleq \sum_{i=1}^{I} \mathbf{z}_i^k + \sum_{(j,i)\in\mathcal{E}} (\boldsymbol{\rho}_{ij}^k - \tilde{\boldsymbol{\rho}}_{ij}^k). \tag{11}$$

We show next that $\mathfrak{m}_z^{k+1} = \mathfrak{m}_z^k = \cdots = \mathfrak{m}_z^0 = \sum_{i=1}^I \mathbf{z}_i^0$. Since agent i^k triggers $k \to k+1$, it is sufficient to show

$$\begin{aligned}
&\mathbf{z}_{i^k}^{k+1} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k+1} - \tilde{\boldsymbol{\rho}}_{i^kj}^{k+1}) + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} (\boldsymbol{\rho}_{ji^k}^{k+1} - \tilde{\boldsymbol{\rho}}_{ji^k}^{k+1}) \\
&= \mathbf{z}_{i^k}^{k} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k} - \tilde{\boldsymbol{\rho}}_{i^kj}^{k}) + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} (\boldsymbol{\rho}_{ji^k}^{k} - \tilde{\boldsymbol{\rho}}_{ji^k}^{k}).
\end{aligned} \tag{12}$$

Using (8)-(10), we can write

$$\begin{aligned}
&\mathbf{z}_{i^k}^{k+1} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k+1} - \tilde{\boldsymbol{\rho}}_{i^kj}^{k+1}) + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} (\boldsymbol{\rho}_{ji^k}^{k+1} - \tilde{\boldsymbol{\rho}}_{ji^k}^{k+1}) \\
&\overset{(a)}{=} a_{i^ki^k}\mathbf{z}_{i^k}^{k+\frac{1}{2}} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k} - \boldsymbol{\rho}_{i^kj}^{k-d_j^k}) \\
&\quad + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} (\boldsymbol{\rho}_{ji^k}^{k} + a_{ji^k}\mathbf{z}_{i^k}^{k+\frac{1}{2}} - \tilde{\boldsymbol{\rho}}_{ji^k}^{k}) \\
&\overset{(b)}{=} \mathbf{z}_{i^k}^{k+\frac{1}{2}} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k} - \boldsymbol{\rho}_{i^kj}^{k-d_j^k}) + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} (\boldsymbol{\rho}_{ji^k}^{k} - \tilde{\boldsymbol{\rho}}_{ji^k}^{k}) \\
&\overset{(8)}{=} \mathbf{z}_{i^k}^{k} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k-d_j^k} - \tilde{\boldsymbol{\rho}}_{i^kj}^{k}) \\
&\quad + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k} - \boldsymbol{\rho}_{i^kj}^{k-d_j^k}) + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} (\boldsymbol{\rho}_{ji^k}^{k} - \tilde{\boldsymbol{\rho}}_{ji^k}^{k}) \\
&= \mathbf{z}_{i^k}^{k} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} (\boldsymbol{\rho}_{i^kj}^{k} - \tilde{\boldsymbol{\rho}}_{i^kj}^{k}) + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} (\boldsymbol{\rho}_{ji^k}^{k} - \tilde{\boldsymbol{\rho}}_{ji^k}^{k})
\end{aligned} \tag{13}$$

where in (a) we used i) (9)-(10), ii) $\boldsymbol{\rho}_{i^kj}^{k+1} = \boldsymbol{\rho}_{i^kj}^{k}$, for all $j \in \mathcal{N}_{i^k}^{\text{in}}$, and iii) $\tilde{\boldsymbol{\rho}}_{ji^k}^{k+1} = \tilde{\boldsymbol{\rho}}_{ji^k}^{k}$, for all $j \in \mathcal{N}_{i^k}^{\text{out}}$; and in (b), we used $a_{i^ki^k} + \sum_{j\in\mathcal{N}_{i^k}^{\text{out}}} a_{ji^k} = 1$.

The mass preservation property above ensures that, if a consensus is reached, i.e., $\lim_{k\to\infty} \mathbf{z}_i^k/\phi_i^k = \mathbf{c}^\infty$ for all $i \in \mathcal{V}$, then it must be $\mathbf{c}^\infty = (1/I)\cdot\sum_{i=1}^I \mathbf{z}_i^0$. This will be formally proved in Theorem 3.2 for the more general P-ASY-SUM-PUSH algorithm, introduced in Sec. III-A.

In Table 2, we summarize the main update of the agent that triggers the generic iteration $k \to k+1$ in a functional form; this will be used in Sec. III-A as a building block of the P-ASY-SUM-PUSH. Note that in the update of the $\mathbf{z}$-variable we added a (possible) perturbation, denoted by $\boldsymbol{\epsilon} \in \mathbb{R}^n$, which will be used in the next section to build a more general algorithm.

A. Perturbed-ASY-Sum-Push (P-ASY-SUM-PUSH)

We are now ready to introduce P-ASY-SUM-PUSH, which serves as a unified algorithmic framework to accomplish several distributed tasks over digraphs in an asynchronous manner, such as solving the average consensus problem and tracking the average of agents' time-varying signals (e.g., the sum of the gradients). More importantly, P-ASY-SUM-PUSH will be the building block of the main algorithm of the paper–ASY-SONATA (cf. Sec. IV).

TABLE 2: The functional $\mathcal{F}$

function $\mathcal{F}(i, k, (\boldsymbol{\rho}_{ij})_{j\in\mathcal{N}_i^{\text{in}}}, (\sigma_{ij})_{j\in\mathcal{N}_i^{\text{in}}}, \boldsymbol{\epsilon})$

(S.1) Sum:

$$\mathbf{z}_i^{k+\frac{1}{2}} = \mathbf{z}_i^k + \sum_{j\in\mathcal{N}_i^{\text{in}}} \left(\boldsymbol{\rho}_{ij} - \tilde{\boldsymbol{\rho}}_{ij}^k\right) + \boldsymbol{\epsilon};$$
$$\phi_i^{k+\frac{1}{2}} = \phi_i^k + \sum_{j\in\mathcal{N}_i^{\text{in}}} \left(\sigma_{ij} - \tilde{\sigma}_{ij}^k\right).$$

(S.2) Push:

$$\mathbf{z}_i^{k+1} = a_{ii}\,\mathbf{z}_i^{k+\frac{1}{2}}, \quad \phi_i^{k+1} = a_{ii}\phi_i^{k+\frac{1}{2}};$$
$$\boldsymbol{\rho}_{ji}^{k+1} = \boldsymbol{\rho}_{ji}^k + a_{ji}\,\mathbf{z}_i^{k+\frac{1}{2}}, \quad \forall j \in \mathcal{N}_i^{\text{out}};$$
$$\sigma_{ji}^{k+1} = \sigma_{ji}^k + a_{ji}\,\phi_i^{k+\frac{1}{2}}, \quad \forall j \in \mathcal{N}_i^{\text{out}}$$

(S.3) Mass-Buffer Update:

$$\tilde{\boldsymbol{\rho}}_{ij}^{k+1} = \boldsymbol{\rho}_{ij}, \quad \tilde{\sigma}_{ij}^{k+1} = \sigma_{ij}, \quad \forall j \in \mathcal{N}_i^{\text{in}}.$$

return $\mathbf{z}_i^{k+1}/\phi_i^{k+1}$.

We introduce a "global view" of the algorithm, which describes the asynchronous actions performed locally by the agents (cf. Table 2) through the dynamics of the "global state" of the system, defined by the tuple $G \triangleq (\mathbf{z}, \phi, \boldsymbol{\rho}, \sigma, \tilde{\boldsymbol{\rho}}, \tilde{\sigma})$. By doing so, one can capture in a unified, abstract model several computational architectures/systems and asynchronous modus operandi. A global iteration clock (not known to the agents) is introduced: $k \to k+1$ is triggered when one agent i^k performs its "push" operation, using possibly delayed information $\boldsymbol{\rho}_{i^kj}^{k-d_j^k}$, $j \in \mathcal{N}_{i^k}^{\text{in}}$, thus generating the vectors $\mathbf{z}_{i^k}^{k+1}$, $\phi_{i^k}^{k+1}$ $(\boldsymbol{\rho}_{ji^k}^{k+1}, \sigma_{ji^k}^{k+1})_{j\in\mathcal{N}_{i^k}^{\text{out}}}$, $(\tilde{\boldsymbol{\rho}}_{i^kj}^{k+1}, \tilde{\sigma}_{i^kj}^{k+1})_{j\in\mathcal{N}_{i^k}^{\text{in}}}$, which completely defines the new state of the system G^{k+1}. Note that the way agent i^k forms its own estimates $\boldsymbol{\rho}_{i^kj}^{k-d_j^k}$ is *immaterial* to the description of the algorithm. Therefore, the update $G^k \to G^{k+1}$ is fully defined, once i^k and $\mathbf{d}^k \triangleq (d_j^k)_{j\in\mathcal{N}_{i^k}^{\text{in}}}$ are given. The P-ASY-SUM-PUSH is summarized in Algorithm 1.

The following comments are in order. The selection $(i^k, \mathbf{d}^k)$ in Step 1 is not performed by anyone; it is instead an *a-posteriori* description of agents' actions: All agents act asynchronously and continuously; the agent completing the "push" step and updating its own variables triggers *retrospectively* the iteration counter $k \to k+1$ and determines the pair $(i^k, \mathbf{d}^k)$ along with all quantities involved in the other steps.

The meaning of Step 2 is the following: agent i^k maintains a local variable τ_{i^kj} that keeps track of the "age" (generated time) of the ρ-variable it has used, initialized to be zero and recursively updated according to (14). If $k - d_j^k$ is larger than τ_{i^kj}, indicating that the received ρ-variable is newer, agent

Algorithm 1 P-ASY-SUM-PUSH (Global View)

Data: $\mathbf{z}_i^0 \in \mathbb{R}^n$, $\phi_i^0 = 1$, $\tilde{\boldsymbol{\rho}}_{ij}^0 = 0$, $\tilde{\sigma}_{ij}^0 = 0$, $\tau_{ij}^{-1} = -D$, for all $j \in \mathcal{N}_i^{\text{in}}$ and $i \in \mathcal{V}$; $\sigma_{ij}^t = 0$ and $\boldsymbol{\rho}_{ij}^t = 0$, for all $t = -D, \dots, 0$; and $\{\boldsymbol{\epsilon}^k\}_{k\in\mathbb{N}_+}$. Set $k = 0$.

While: a termination criterion is not met **do**

(S.1) Pick $(i^k, \mathbf{d}^k)$;

(S.2) Set (purge out the old information):

$$\tau_{i^k j}^k = \max\left(\tau_{i^k j}^{k-1}, k - d_j^k\right), \quad \forall j \in \mathcal{N}_{i^k}^{\text{in}}; \tag{14}$$

(S.3) Update the variables performing

$$\mathbf{y}_{i^k}^{k+1} = \mathcal{F}(i^k, k, (\boldsymbol{\rho}_{i^k j}^{\tau_{i^k j}^k})_{j\in\mathcal{N}_{i^k}^{\text{in}}}, (\sigma_{i^k j}^{\tau_{i^k j}^k})_{j\in\mathcal{N}_{i^k}^{\text{in}}}, \boldsymbol{\epsilon}^k);$$

(S.4) Untouched state variables shift to state $k+1$ while keeping the same value; $k \leftarrow k+1$.

i^k accepts $\boldsymbol{\rho}_{i^k j}^{k-d_j^k}$ and updates $\tau_{i^k j}$ as $k - d_j^k$; otherwise, the variable will be discarded and $\tau_{i^k j}$ is not changed. Note that (14) can be performed without any coordination. It is sufficient that each agent attaches a time-stamp to its produced information reflecting it local timing counter.

Convergence is established under the following assumptions, and it is stated in Theorem 3.2.

Assumption 3.1 (On the asynchronous model): Suppose:

(i) $\exists\ 0 < T < \infty$ such that $\cup_{t=k}^{k+T-1} i^t = \mathcal{V}$, for all $k \in \mathbb{N}_+$;

(ii) $\exists\ 0 < D < \infty$ such that $0 \le d_j^k \le D$, for all $j \in \mathcal{N}_{i^k}^{\text{in}}$ and $k \in \mathbb{N}_+$. □

Theorem 3.2: Let $\{(\mathbf{y}_i^k)_{i=1}^I, (\boldsymbol{\rho}_{ij}^k, \tilde{\boldsymbol{\rho}}_{ij}^k)_{(j,i)\in\mathcal{E}}\}_{k\in\mathbb{N}_+}$ be the sequence generated by Algorithm 1, under Assumption 3.1, and with $\mathbf{A} \triangleq (a_{ij})_{i,j=1}^I$ satisfying Assumption 2.4(i),(iii). Let $\mathfrak{m}_z^k$ be defined in (11); we have $\mathfrak{m}_z^k = \sum_{i=1}^I \mathbf{z}_i^0 + \sum_{t=0}^{k-1} \boldsymbol{\epsilon}^t$. There exist constants $\rho \in (0,1)$ and $C_1 > 0$, such that, for all $i \in \mathcal{V}$ and $k \ge (2\,I-1)\cdot T + I \cdot D - 1$,

$$\left\|\mathbf{y}_i^{k+1} - (1/I)\cdot \mathfrak{m}_z^{k+1}\right\| \le C_1 \left(\rho^k \|\mathbf{z}^0\| + \sum_{l=0}^{k} \rho^{k-l} \|\boldsymbol{\epsilon}^l\|\right). \tag{15}$$

Proof: The proof is involved and can be found in [38], along with the expressions of the above constants. ■

Discussion: Several comments are in order.

1) On the asynchronous model: Algorithm 1 represents a gamut of asynchronous *parallel* schemes and architectures, all captured in an abstract and unified way by the mechanism of generation of the indices i^k and delay vectors $\mathbf{d}^k$, which the agents *need not know*. The only conditions to be satisfied by $(i^k, \mathbf{d}^k)$ are in Assumption 3.1: (i) controls the frequency of the updates whereas (ii) limits the age of the old information used in the computations. These assumptions are quite mild. For instance, (i) is automatically satisfied if each agent wakes up and performs an update whenever some internal clock ticks, without the need of any central clock or coordination with the others. Assumption 3.1(ii) imposes some conditions on the communications: the information used by any agent is outdated by at most D units (with D finite but arbitrarily large). This however does not enforce *a-priori* any specific protocol (on the activation/idle time/communication). For instance, i) agents need not perform the actions in Table 2 sequentially or inside the same activation; ii) executing the "push" step does not mean that agents must broadcast their new variables in the same activation; this would incur in a delay (or packet loss) in the communication.

2) Beyond average consensus: By choosing properly the perturbation signal $\boldsymbol{\epsilon}^k$, P-ASY-SUM-PUSH can solve more general problems than the plain average consensus. Some examples are discussed next.

(i) Error free: $\boldsymbol{\epsilon}^k = \mathbf{0}$. P-ASY-SUM-PUSH solves the average consensus problem and (15) reads

$$\left\|\mathbf{y}_i^{k+1} - (1/I)\cdot\sum_{i=1}^I \mathbf{z}_i^0\right\| \le C_1\,\rho^k\,\|\mathbf{z}^0\|.$$

(ii) Vanishing error: $\lim_{k\to\infty}\|\boldsymbol{\epsilon}^k\| = 0$. Using [30, Lemma 7(a)], (15) reads $\lim_{k\to\infty}\|\mathbf{y}_i^{k+1} - \mathfrak{m}_z^{k+1}\| = 0$.

(iii) Asynchronous tracking. Each agent i owns a (time-varying) signal $\{\mathbf{u}_i^k\}_{k\in\mathbb{N}_+}$; the average tracking problem consists in asymptotically track the average signal $\bar{\mathbf{u}}_i^k \triangleq (1/I)\cdot\sum_{i=1}^I \mathbf{u}_i^k$, that is,

$$\lim_{k\to\infty} \|\mathbf{y}_i^{k+1} - \bar{\mathbf{u}}_i^{k+1}\| = 0, \quad \forall i \in \mathcal{V}. \tag{16}$$

Under mild conditions on the signal, this can be accomplished in a distributed and asynchronous fashion, using P-ASY-SUM-PUSH, with the following setting: $\mathbf{z}_i^0 = \mathbf{u}_i^0$, for all $i \in \mathcal{V}$; $\boldsymbol{\epsilon}^k = \mathbf{u}_{i^k}^{k+1} - \tilde{\mathbf{u}}_{i^k}^k$, with

$$\tilde{\mathbf{u}}_i^{k+1} = \begin{cases} \mathbf{u}_i^{k+1} & \text{if } i = i^k; \\ \tilde{\mathbf{u}}_i^k & \text{otherwise;} \end{cases}$$

and $\tilde{\mathbf{u}}_i^0 = \mathbf{u}_i^0$. In this setting, (15) holds, with $\mathfrak{m}_z^{k+1} = \sum_{i=1}^I \tilde{\mathbf{u}}_i^{k+1}$. Furtheremore, if $\lim_{k\to\infty}\sum_{i=1}^I \|\mathbf{u}_i^{k+1} - \mathbf{u}_i^k\| = 0$, one can show that (16) holds. This instance of P-ASY-SUM-PUSH will be used in Sec. IV to perform asynchronous gradient tracking inside ASY-SONATA.

Remark 3.3 (Comparison with [6], [31], [32]): As already mentioned, coding the information throughout counter variables (like $\boldsymbol{\rho}$-, $\sigma-$, $\tilde{\boldsymbol{\rho}}$-, and $\tilde{\sigma}$-variables in our scheme) was first introduced in [31] to design a synchronous average consensus algorithm robust to packet losses. In [32], this scheme was extended to deal with uncoordinated (deterministic) agents' activations whereas [6] built on [32] to design, in the same setting, a distributed Newton-Rapshon algorithm. There are several important differences between P-ASY-SUM-PUSH and the aforementioned schemes, namely: i) none of them can deal with *delays but packet losses*; ii) [31] is *synchronous*; and iii) [6], [32] are not *parallel* schemes, as at each iteration only one agent is allowed to wake up and transmit information to its neighbors. For instance, [6], [32] cannot model synchronous parallel (Jacobi) updates.

IV. ASYNCHRONOUS SONATA (ASY-SONATA)

We are ready now to introduce our distributed asynchronous algorithmic framework–ASY-SONATA. The algorithm combines SONATA (cf. Sec. III) with P-ASY-SUM-PUSH (cf. Sec. III-A), the latter replacing the synchronous tracking scheme (2)-(4). The "global view" of the scheme is given in Algorithm 2. Note that, each agent i now, in addition to its x- and y-variables, also owns the mass-counters $((\boldsymbol{\rho}_{ji}^k)_{j\in\mathcal{N}_i^{\text{out}}}, (\sigma_{ji}^k)_{j\in\mathcal{N}_i^{\text{out}}})$, and the buffers $((\tilde{\boldsymbol{\rho}}_{ij}^k)_{j\in\mathcal{N}_i^{\text{in}}}, (\tilde{\sigma}_{ij}^k)_{j\in\mathcal{N}_i^{\text{in}}})$, which are used to employ gradient tracking in an asynchronous fashion. For notational simplicity and without loss of generality, we assumed that the v- and y- variables are subject to the same delays (e.g., they are transmitted within the same packet); same convergence results hold if different delays are considered [38].

In ASY-SONATA, agents continuously and with no coordination perform: (Step 1) their local computations, possibly using an out-of-sync estimate $\mathbf{y}_{i^k}^k$ of the average gradient; (Step 2) a consensus step on the x variables, using possibly outdated information $\mathbf{v}_j^{k-d_j^k}$ from their in-neighbors; and (Step 3) gradient tracking to update the local estimate $\mathbf{y}_{i^k}^k$ by employing the functional $\mathcal{F}$ (cf. Table 2), based on the current mass counters $((\boldsymbol{\rho}_{i^kj}^{k-d_j^k})_{j\in\mathcal{N}_{i^k}^{\text{in}}}, (\sigma_{i^kj}^{k-d_j^k})_{j\in\mathcal{N}_{i^k}^{\text{in}}})$. In (17), a step-size is used. We discuss next the convergence properties of the scheme, using either a constant step-size or diminishing, uncoordinated ones.

Algorithm 2 ASY-SONATA (Global View)

Data: For all agent i and $\forall j \in \mathcal{N}_i^{\text{in}}$, $\mathbf{x}_i^0 \in \mathbb{R}^n$, $\mathbf{z}_i^0 = \nabla f_i(\mathbf{x}_i^0)$, $\phi_i^0 = 1$, $\tilde{\boldsymbol{\rho}}_{ij}^0 = 0$, $\tilde{\sigma}_{ij}^0 = 0$, $\tau_{ij}^{-1} = -D$. And for $t = -D, -D+1, \ldots, 0$, $\boldsymbol{\rho}_{ij}^t = 0$, $\sigma_{ij}^t = 0$, $\mathbf{v}_i^t = 0$. Set $k = 0$.

While: a termination criterion is not met **do**

(S.1) Pick $(i^k, \mathbf{d}^k)$;

(S.2) Set:

$$\tau_{i^kj}^k = \max(\tau_{i^kj}^{k-1}, k - d_j^k), \quad \forall j \in \mathcal{N}_{i^k}^{\text{in}}.$$

(S.3) Local Descent:

$$\mathbf{v}_{i^k}^{k+1} = \mathbf{x}_{i^k}^k - \gamma^k \mathbf{y}_{i^k}^k. \tag{17}$$

(S.4) Consensus:

$$\mathbf{x}_{i^k}^{k+1} = w_{i^ki^k}\mathbf{v}_{i^k}^{k+1} + \sum_{j\in\mathcal{N}_{i^k}^{\text{in}}} w_{i^kj}\mathbf{v}_j^{\tau_{i^kj}^k}.$$

(S.5) Gradient Tracking:

$$\boldsymbol{\epsilon}^k = \nabla f_{i^k}(\mathbf{x}_{i^k}^{k+1}) - \nabla f_{i^k}(\mathbf{x}_{i^k}^k)$$

$$\mathbf{y}_{i^k}^{k+1} = \mathcal{F}(i^k, k, \{\boldsymbol{\rho}_{i^kj}^{\tau_{i^kj}^k}\}_{j\in\mathcal{N}_{i^k}^{\text{in}}}, \{\sigma_{i^kj}^{\tau_{i^kj}^k}\}_{j\in\mathcal{N}_{i^k}^{\text{in}}}, \boldsymbol{\epsilon}^k).$$

(S.6) Untouched state variables shift to state $k+1$ while keeping the same value; $k \leftarrow k+1$.

A. Constant Step-size

Convergence under a constant step-size is given in Theorem 4.1 and Theorem 4.2 below, for the case of strongly convex and (non)convex function F, respectively. The proof of the two theorems is quite involved and can be found in [38]. We use the following merit function

$$M_F(\mathbf{x}^k) \triangleq \max\{\|\nabla F(\bar{\mathbf{x}}^k)\|^2, \|\mathbf{x}^k - \mathbf{1}_I \otimes \bar{\mathbf{x}}^k\|^2\}, \tag{18}$$

where $\mathbf{x}^k \triangleq [\mathbf{x}_1^{k\top}, \cdots, \mathbf{x}_I^{k\top}]^\top$ and $\bar{\mathbf{x}}^k \triangleq (1/I)\cdot\sum_{i=1}^I \mathbf{x}_i^k$. Note that M_F is a valid merit function, since it is continuous and $M_F(\mathbf{x}) = 0$ if and only if all $\mathbf{x}_i$'s are consensual and optimal (resp. stationary solutions).

Theorem 4.1 (Linear convergence): Consider Problem (P) under Assumption 2.2, and let $\mathbf{x}^\star$ denote its unique solution. Let $\{(\mathbf{x}_i^k)_{i=1}^I\}_{k\in\mathbb{N}_+}$ be the sequence generated by Algorithm 2, under Assumption 3.1, and with weight-matrices $\mathbf{W}$ and $\mathbf{A}$ satisfying Assumption 2.4. Then, there exists a constant $\bar{\gamma}_1 > 0$ such that if $\gamma^k \equiv \gamma \le \bar{\gamma}_1$, it holds

$$\|\mathbf{x}_i - \mathbf{x}^\star\| = \mathcal{O}(\lambda^k), \quad \forall i \in \mathcal{V}, \tag{19}$$

for some $\lambda \in (0,1)$. □

Theorem 4.2 (Sublinear convergence): Consider Problem (P) under Assumption 2.1 (thus possibly nonconvex). Let $\{(\mathbf{x}_i^k)_{i=1}^I\}_{k\in\mathbb{N}_+}$ be the sequence generated by Algorithm 2, in the same setting of Theorem 4.1. Given $\delta > 0$, let T_δ be the first iteration $k \in \mathbb{N}_+$ such that $M_F(\mathbf{x}^k) \le \delta$. Then, there exists a $\bar{\gamma}_2 > 0$, such that if $\gamma^k \equiv \gamma \le \bar{\gamma}_2$, $T_\delta = \mathcal{O}(1/\delta)$.□

The expression of the constants in the theorems above can be found in [38].

Theorem 4.1 states that both consensus and optimization errors of the sequence generated by ASY-SONATA vanish geometrically. Therefore, ASY-SONATA matches the performance of a centralized gradient method in a distributed, asynchronous computing environment. We are not aware of any other scheme enjoying such a property in the considered setting. Note that ASY-SONATA is *globally* convergent *regardless of the initialization*. This is a major difference with respect to the distributed algorithm proposed in [6] (also employing a robustification of the push-sum consensus, cf. Remark 3.3). Convergence therein is established assuming that all agents initialize their local copies to be almost consensual and in a neighborhood of the optimal solution.

Finally, Theorem 4.2 shows that for general, possibly nonconvex instances of Problem (P), both consensus and optimization errors of the sequence generated by ASY-SONATA vanish at $\mathcal{O}(1/\delta)$ sublinear rate.

B. Uncoordinated diminishing step-sizes

While the use of a constant step-size is appealing to obtain strong convergence rate results, obtaining in a distributed setting its upper-bound expression, as required in Theorems 4.1 and 4.2, is not practical. In fact, such a value depends on global network and optimization parameters, not available at the agents' side. Furthermore, the theoretical values are quite conservative, meaning that they would lead to slow convergence in practice. This naturally suggests the

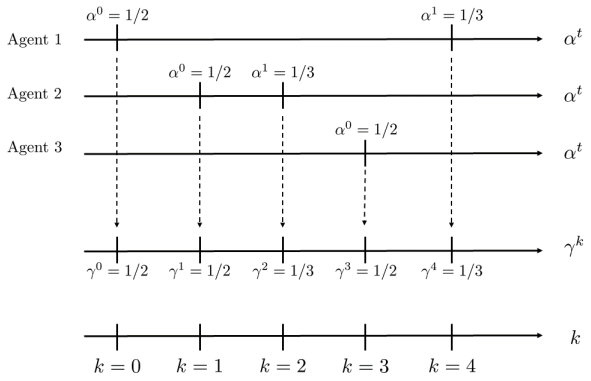

Fig. 1: Example of construction of the global step-size sequence $\{\gamma^k\}_k$ in (17) from the local agents' ones $\{\alpha^t\}_t = \{1/2, 1/3, \ldots\}$.

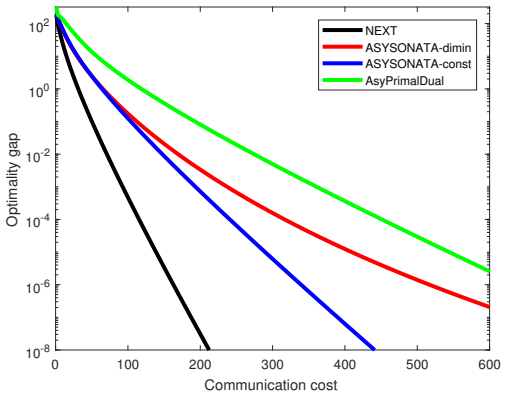

Fig. 2: LS problem: Distance from optimality J^k versus communication cost.

use of a diminishing step-size strategy. However, because of the distributed nature of the system, one cannot simply assume that the sequence $\{\gamma^k\}_{k\in\mathbb{N}_+}$ in (17) is a classical diminishing step-size sequence. In fact, this would require each agent to know the global iteration counter k, which is impossible. Inspired by [39], we assume instead that each agent, *independently* and with *no coordination* with the others, draws the step-size from a local sequence $\{\alpha^t\}_{t\in\mathbb{N}_+}$, according to its local clock. The sequence $\{\gamma^k\}_{k\in\mathbb{N}_+}$ will be thus the result of the "uncoordinated samplings" of the local out-of-sync sequences $\{\alpha^t\}_{t\in\mathbb{N}_+}$. Fig. 1 shows an example of how the resulting sequence $\{\gamma^k\}_{k\in\mathbb{N}_+}$ is built, with three agents and $\{\alpha^t\}_{t\in\mathbb{N}_+} = \{1/2, 1/3, \ldots\}$.

The next theorem shows that in this setting, ASY-SONATA converges sub-linearly for both convex and nonconvex instances of (P)–the proof can be found in [38].

Theorem 4.3: Consider (P) under Assumption 2.1 (thus possibly nonconvex). Let $\{(\mathbf{x}_i^k)_{i=1}^I\}_{k\in\mathbb{N}_+}$ be the sequence generated by Algorithm 2, in the same setting of Theorem 4.1, but with the agents using a local step-size sequence $\{\alpha^t\}_{t\in\mathbb{N}_+}$ satisfying $\alpha^t \downarrow 0$ and $\sum_{t=0}^{\infty}\alpha^t = \infty$. Given $\delta > 0$, let T_δ be the first iteration $k \in \mathbb{N}_+$ such that $M_F(\mathbf{x}) \le \delta$. Then

$$T_\delta \le \inf\Big\{k \ge 0 \,\Big|\, \sum_{t=0}^{k}\gamma^t \ge c/\delta\Big\},$$

where c is a positive constant (see [38]). □

V. Numerical Results

We test ASY-SONATA on the Least Squares (LS) problem, a strongly convex instance of Problem (P). We compare it with the asynchronous primal-dual based method in [8] (referred to as ASY-PrimalDual). As benchmark, we also apply NEXT [30] in the synchronous setting.

In the LS problem, each agent i estimates an unknown signal $\mathbf{x}_0 \in \mathbb{R}^n$ through linear measurements $\mathbf{b}_i = \mathbf{M}_i\mathbf{x}_0 + \mathbf{n}_i$, where $\mathbf{M}_i \in \mathbb{R}^{d_i\times n}$ is the sensing matrix, and $\mathbf{n}_i \in \mathbb{R}^{d_i}$ is the additive noise. The LS problem can be written in the form of (P), with each $f_i(\mathbf{x}) = \|\mathbf{M}_i\mathbf{x} - \mathbf{b}_i\|^2$. We set $n = 200$ and fix $\mathbf{x}_0$ with its elements being i.i.d. random variables drawn from the standard normal distribution. Each agent i takes $d_i = 30$ observations. For each $\mathbf{M}_i$, we firstly generate all the elements as i.i.d. random variables drawn from the standard normal distribution, and then normalize the matrix by multiplying it with the reciprocal of its spectral norm. The components of the additive noise $\mathbf{n}_i$ are i.i.d., Gaussian distributed, with zero mean and variance equal to 0.04.

We simulate a network of $I = 30$ agents; the traveling time of each packet is an integer taken uniformly from the interval $[0, 40]$. Since both ASY-PrimalDual and NEXT can only be applied to undirected graphs, in our experiment, we consider such a setting. A graph is firstly generated according to the Erdos-Renyi model, with parameter $p = 0.3$ (which represents the probability of having an edge between any two nodes), and is selected if its algebraic connectivity is strictly greater than 2. A compatible doubly stochastic weight matrix is then generated by the Metropolis-Hasting rule. We test ASY-SONATA using both constant and uncoordinated diminishing step-size, with the following tuning. In ASY-SONATA (resp. NEXT), the constant step-size is set to $\gamma = 0.84$ (resp. $\gamma = 0.9$) whereas the local diminishing step-size of each agent are chosen as $\alpha^{t+1} = \alpha^t\,(1 - \mu\,\alpha^t)$, with $\alpha^0 = 0.9$ and $\mu = 0.003$. In ASY-PrimalDual, the step-size is set to be $\alpha = 0.62$ and $\eta = 0.9$ (see [8] for a definition of this quantities). These choices lead to the best practical performance for the algorithms.

In Fig. 2, we plot the optimality gap defined as $\frac{1}{I}\sum_{i=1}^I \|\mathbf{x}_i^k - \mathbf{x}^\star\|_2^2$ achieved by all the algorithms versus the communication cost, where $\mathbf{x}^\star$ is the unique solution of the LS. The curves are averaged over 100 Monte-Carlo simulations with different graph instantiations. The communication cost is counted as the number of scalars sent by all agents in the network at each iteration. It is not difficult to check that, at each iteration, in ASY-SONATA, each agent transmits $(2n+1)I$ scalars (namely: $\mathbf{v}$, $\boldsymbol{\rho}$, and σ) whereas in ASY-PrimalDual the number of scalars to be transmitted is $(I + \frac{|\mathcal{E}|}{2})\,n$ (both node and edge variables). The figure shows that ASY-SONATA converges faster than ASY-PrimalDual; equivalently, it reaches the same solution accuracy requiring less communications. Also, ASY-SONATA achieves linear convergence rate and, quite remarkably, it exhibits performance close to the synchronous scheme, NEXT.

VI. Conclusions

We proposed ASY-SONATA, a distributed asynchronous algorithmic framework for convex and nonconvex (unconstrained, smooth) multi-agent problems, over digraphs. The algorithm is robust against uncoordinated agents' activation and (communication/computation) (time-varying) delays. When employing a constant step-size, ASY-SONATA achieves a linear rate for strongly convex problems–matching the rate of a centralized gradient algorithm–and sublinear rate for (non)convex problems. Sublinear rate is also established when agents employ uncoordinated diminishing step-sizes, which is more realistic in a distributed setting. To the best of our knowledge, ASY-SONATA is the first distributed algorithm enjoying the above properties, in the general asynchronous setting described in the paper.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] A. Nedić, "Asynchronous broadcast-based convex optimization over a network," *IEEE Trans. Automat. Contr.*, vol. 56, no. 6, pp. 1337–1351, 2011.

[3] X. Zhao and A. H. Sayed, "Asynchronous adaptation and learning over networks–Part I/Part II/Part III: Modeling and stability analysis/Performance analysis/Comparison analysis," *IEEE Trans. Signal Process.*, vol. 63, no. 4, pp. 811–858, 2015.

[4] S. Kumar, R. Jain, and K. Rajawat, "Asynchronous optimization over heterogeneous networks via consensus admm," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 3, no. 1, pp. 114–129, 2017.

[5] M. Eisen, A. Mokhtari, and A. Ribeiro, "Decentralized quasi-newton methods," *IEEE Trans. Signal Process.*, vol. 65, no. 10, pp. 2613–2628, 2017.

[6] N. Bof, R. Carli, G. Notarstefano, L. Schenato, and D. Varagnolo, "Newton-raphson consensus under asynchronous and lossy communications for peer-to-peer networks," *arXiv:1707.09178*, 2017.

[7] Z. Peng, Y. Xu, M. Yan, and W. Yin, "Arock: an algorithmic framework for asynchronous parallel coordinate updates," *SIAM J. Sci. Comput.*, vol. 38, no. 5, pp. A2851–A2879, 2016.

[8] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized consensus optimization with asynchrony and delays," *IEEE Trans. Signal Inf. Process. Netw.*, vol. PP, no. 99, 2017.

[9] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Trans. Automat. Contr.*, vol. 31, no. 9, pp. 803–812, 1986.

[10] J. Liu and S. J. Wright, "Asynchronous stochastic coordinate descent: Parallelism and convergence properties," *SIAM J. Optim.*, vol. 25, no. 1, pp. 351–376, 2015.

[11] L. Cannelli, F. Facchinei, V. Kungurtsev, and G. Scutari, "Asynchronous parallel algorithms for nonconvex big-data optimization–Part I & Part II: Model and convergence & Complexity and numerical results," *arXiv:1607.04818 & arXiv:1701.04900*, 2016.

[12] F. Niu, B. Recht, C. Re, and S. J. Wright, "Hogwild: a lock-free approach to parallelizing stochastic gradient descent," in *Proc. of NIPS 2011*, pp. 693–701.

[13] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. of NIPS 2015*, pp. 2719–2727.

[14] A. Mokhtari, A. Koppel, and A. Ribeiro, "A class of parallel doubly stochastic algorithms for large-scale learning," *arXiv:1606.04991*, 2016.

[15] A. Nedić, D. P. Bertsekas, and V. S. Borkar, "Distributed asynchronous incremental subgradient methods," *Stud. Comput. Math.*, vol. 8, no. C, pp. 381–407, 2001.

[16] Z. Huo and H. Huang, "Asynchronous mini-batch gradient descent with variance reduction for non-convex optimization," in *Proc. of AAAI 2017*, 2017, pp. 2043–2049.

[17] I. Notarnicola and G. Notarstefano, "Asynchronous distributed optimization via randomized dual proximal gradient," *IEEE Trans. Automat. Contr.*, vol. 62, no. 5, pp. 2095–2106, 2017.

[18] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Convergence of asynchronous distributed gradient methods over stochastic networks," *IEEE Trans. Automat. Contr.*, vol. 63, no. 2, pp. 434–448, 2017.

[19] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *Proc. of CDC 2013*, pp. 3671–3676.

[20] E. Wei and A. Ozdaglar, "On the $o(1/k)$ convergence of asynchronous distributed alternating direction method of multipliers," in *Proc. of GlobalSIP 2013*, pp. 551–554.

[21] P. Bianchi, W. Hachem, and F. Iutzeler, "A coordinate descent primal-dual algorithm and application to distributed asynchronous optimization," *IEEE Trans. Automat. Contr.*, vol. 61, no. 10, pp. 2947–2957, 2016.

[22] H. Wang, X. Liao, T. Huang, and C. Li, "Cooperative distributed optimization in multiagent networks with delays," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 45, no. 2, pp. 363–369, 2015.

[23] J. Li, G. Chen, Z. Dong, and Z. Wu, "Distributed mirror descent method for multi-agent optimization with delay," *Neurocomputing*, vol. 177, pp. 643–650, 2016.

[24] K. I. Tsianos and M. G. Rabbat, "Distributed dual averaging for convex optimization under communication delays," in *Proc. of ACC 2012*, pp. 1067–1072.

[25] ——, "Distributed consensus and optimization under communication delays," in *Proc. of Allerton 2011*, pp. 974–982.

[26] P. Lin, W. Ren, and Y. Song, "Distributed multi-agent optimization subject to nonidentical constraints and communication delays," *Automatica*, vol. 65, pp. 120–131, 2016.

[27] T. T. Doan, C. L. Beck, and R. Srikant, "Impact of communication delays on the convergence rate of distributed optimization algorithms," *arXiv:1708.03277*, 2017.

[28] P. Di Lorenzo and G. Scutari, "Distributed nonconvex optimization over networks," in *Proc. of IEEE CAMSAP*, 2015, pp. 229–232.

[29] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes," in *Proc. of CDC 2015*, Osaka, Japan, Dec., pp. 2055–2060.

[30] P. Di Lorenzo and G. Scutari, "Next: In-network nonconvex optimization," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 2, no. 2, pp. 120–136, 2016.

[31] C. N. Hadjicostis, N. H. Vaidya, and A. D. Dominguez-Garcia, "Robust distributed average consensus via exchange of running sums," *IEEE Trans. Automat. Contr.*, vol. 31, no. 6, pp. 1492–1507, 2016.

[32] N. Bof, R. Carli, and L. Schenato, "Average consensus with asynchronous updates and unreliable communication," in *Proc. of the IFAC Word Congress 2017*, pp. 601–606.

[33] Y. Sun, G. Scutari, and D. Palomar, "Distributed nonconvex multiagent optimization over time-varying networks," in *Proc. of Asilomar 2016*. IEEE, pp. 788–794.

[34] G. Scutari and Y. Sun, "Distributed nonconvex constrained optimization over time-varying digraphs," *arXiv:1809.01106*, 2018.

[35] A. Nedić, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM J. Optim.*, vol. 27, no. 4, pp. 2597–2633, 2017.

[36] Y. Sun, A. Daneshmand, and G. Scutari, "Convergence rate of distributed convex and nonconvex optimization methods with gradient tracking," Purdue University, Tech. Rep., 2018.

[37] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. of FOCS 2003*, pp. 482–491.

[38] Y. Tian, Y. Sun, and G. Scutari, "Achieving geometric convergence for distributed asynchronous optimization," *arXiv:1803.10359*, 2018.

[39] L. Cannelli, F. Facchinei, and G. Scutari, "Multi-agent asynchronous nonconvex large-scale optimization," in *Proc. of IEEE CAMSAP 2017*, pp. 1–5.