EURASIP Journal on Advances in Signal Processing

RESEARCH Open Access

Performance evaluation and analysis of distributed multi-agent optimization algorithms with sparsified directed communication

Lidija Fodor*, Dušan Jakovetić, Nataša Krejić, Nataša Krklec Jerinkić and Srđan Škrbić

*Correspondence:
lidija.fodor@dmi.uns.ac.rs
Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia

Abstract

There has been significant interest in distributed optimization algorithms, motivated by applications in Big Data analytics, smart grid, vehicle networks, etc. While there have been extensive theory and theoretical advances, a proportionally small body of scientific literature focuses on numerical evaluation of the proposed methods in actual practical, parallel programming environments. This paper considers a general algorithmic framework of first and second order methods with sparsified communications and computations across worker nodes. The considered framework subsumes several existing methods. In addition, a novel method that utilizes unidirectional sparsified communications is introduced and theoretical convergence analysis is also provided. Namely, we prove R-linear convergence in the expected norm. A thorough empirical evaluation of the methods using Message Passing Interface (MPI) on a High Performance Computing (HPC) cluster is carried out and several useful insights and guidelines on the performance of algorithms and inherent communication-computational trade-offs in a realistic setting are derived.

Keywords: Distributed optimization, High performance computing, Performance evaluation

1 Introduction

Distributed multi-agent optimization is today a mature theoretical area, e.g. [1]. Several first [1–3] and second order [4–6] distributed methods have been proposed, and their theoretical properties have been well understood, e.g., in terms of theoretical iteration-wise convergence rates.

Distributed multi-agent optimization methods have a great potential in various application domains, including distributed machine learning [7], distributed control [8], vehicular networks [9], smart grid [10], etc. Relevant applications have been also demonstrated [11]. However, there is a restricted amount of scientific investigation of distributed multi-agent optimization methods in realistic distributed computational/High Performance Computing (HPC) systems. Carrying out such studies is extremely important as

there is a significant gap between theoretical studies of the methods and actual performance in practical infrastructures. For example, it is very hard to understand the relationship between iteration-wise convergence rate and actual communication and computational costs without empirical evaluation.

In this paper, a thorough and systematic empirical study of a class of distributed multi-agent optimization methods is carried out. All these methods are defined by different variants of sparsification of communications and/or computations along iterations. In more detail, we consider both first and second order methods that exhibit either unidirectional or bidirectional randomized sparsified communications. The considered sparsification strategies involve parameter p_k that represents the probability that a node communicates at iteration k; the quantity p_k is a design parameter that is either increasing, decreasing, or constant. These strategies give rise to a number of methods summarized in Table 1[1]. The studied class of methods subsumes several existing algorithms [12–17] but also includes several methods that have not been studied before, neither numerically nor analytically. Further contribution to the literature body of analytical results is the presented convergence analysis of the FUI method.

The main aims of the empirical evaluation are as follows: 1) to assess real benefits of sparsifying communications and/or computations across nodes, which have been proved to be beneficial theoretically [16]; 2) to compare different alternatives of the sparsification strategies; and 3) to compare unidirectional and bidirectional communication strategies. One of the main motivations for using sparsified, randomized communication is to reduce the amount of time spent on data exchange. The choice of omitting to communicate in some cases can also lead to savings in bandwidth or transmission power of wireless devices, when considering wireless networks. Using randomized communication at the level of algorithm design is a well established topic, where, e.g., gossip [18] is an outstanding example. It is also of interest to explore the case when communication sparsification is not fully determined by the algorithm designer, but instead is dictated by random link failures (e.g., packet dropouts in wireless networks).

The underlying implementation framework is the MPI (Message Passing Interface, [19]) running on an HPC computer cluster, which is a standard and widely adopted computational system.

Table 1 Different alternatives of Algorithm 1

Method	Type	M_i^k	$\xi_{i,j}^k$	p_k	Relevant reference
FBI	First order	I	$z_i^k \cdot z_j^k$	$p_k = 1 - 0.5^k$	[12]
FBD	First order	I	$z_i^k \cdot z_j^k$	$p_k = (k+1)^{-1}$	[13]
FUI	First order	I	z_j^k	$p_k = 1 - 0.5^k$	this paper
FUD	First order	I	z_j^k	$p_k = (k+1)^{-1}$	[14]
FBC	First order	I	1	1	[15]
SBC	Second order	D_i^k	1	1	[5]
SBI	Second order	D_i^k	$z_i^k \cdot z_j^k$	$p_k = 1 - 0.5^k$	[16]
SBD	Second order	D_i^k	$z_i^k \cdot z_j^k$	$p_k = (k+1)^{-1}$	[13, 16]
SUI	Second order	D_i^k	z_j^k	$p_k = 1 - 0.5^k$	this paper
SUD	Second order	D_i^k	z_j^k	$p_k = (k+1)^{-1}$	this paper

[1] Our convention for abbreviating the methods uses a three letter system, where the first letter represents whether the method is first or second order (F or S); the second letter represents the type of the communication (B for bidirectional and U for unidirectional); the third letter represents the communication sparsification type, i.e. the probability used for communication (I for increasing, D for decreasing and C for constant)

The rest of the paper is organized as follows. An overview of the work related to this topic is presented in Subsection 1.1. We briefly describe the optimization model and the algorithmic framework for the Distributed Quasi Newton (DQN) method in Subsection 2.1. The algorithmic framework is described in Subsection 2.2. Convergence analysis of the novel FUI method that uses unidirectional communication is presented in Subsection 2.3. Implementation and infrastructure are described in Subsection 2.4. The simulation setup is described in Subsection 2.5 and the proposed set of methods that fit the introduced algorithm is presented in Subsection 2.6. The results are highlighted in Section 3, organized in the following manner: Subsection 3.1 contains an analysis of different graph types; Subsection 3.2 is dedicated to the analysis of scaling properties of the algorithm; Subsection 3.3 contains an analysis and comparison of execution time, regarding all the considered methods; an analysis and discussion on the effects of communication sparsification is given in Subsections 3.4 and 3.5 is dedicated to the comparison of Algorithm 1 to ADMM (Alternating Direction Method of Multipliers, see [11]). Finally, the conclusions are made in Section 4.

1.1 Related work

There has been a large body of literature on theoretical development of distributed optimization methods. A proportionally much smaller body of scientific literature focuses on testing and evaluation of the methods over actual distributed infrastructures. Existing studies include, e.g., [20], for the dual averaging method, and [11] for the alternating direction method of multipliers. Distributed convex optimization by alternating direction method of multipliers is studied in [11]. A stochastic, efficient quasi-Newton method, using the BFGS update formula, is introduced in [21] in order to take advantage of the curvature information. A fast distributed proximal gradient method is proposed in [22]. An incremental sub-gradient approach, suitable for distributed optimization in networked systems, is presented in [23]. An important aspect in evaluation in distributed optimization is the nature of the network of nodes itself. The effects of this aspect are highlighted in [24].

More recently, there have been works that include MPI-based empirical studies of the methods. In [25] an asynchronous subgradient-push method is proposed and its performance is evaluated on an MPI cluster, whereas in [26] an empirical comparison of several distributed first order methods is given. An exact asynchronous method and its performance analysis using an MPI cluster are presented in [27]. A theoretical and empirical study of communication and computational trade-offs for the distributed dual averaging method is given in [20]. Finally, the focus of [28] is on the distributed dual averaging method with several useful guidelines about practical design and performance of the methods.

With respect to existing studies, this paper differs along several lines. First, it considers a different class of methods with respect to existing empirical studies, as the considered methods include various strategies for communication sparsification (see [12–17]). Second, it provides a novel insights into comparison among different sparsification strategies, as well as the practical benefits with respect to the corresponding always-communicating benchmark. The empirical results show that communication sparsification can lead to significant execution time reductions. To the best of our knowledge, this is the first empirical evaluation reported on the class of algorithms with sparsified communications presented in [16].

Also, a theoretical convergence analysis of the FUI method is carried out in this paper. While [14] also considers unidirectional communications, it studies the specific problem of distributed estimation, which translates into quadratic objective functions and stochastic gradient updates. In contrast, our analysis considers generic strongly convex costs. An important aspect of the framework considered in this paper is that it includes both first and second order methods.

2 Methods

2.1 Optimization and network models

Consider a connected network of n nodes, where each node has access to a convex cost function $f_i : \mathbb{R}^s \to \mathbb{R}$, and assume that f_i is known only by the node i. The goal is to solve the following unconstrained optimization problem

$$\min f(x) := \sum_{i=1}^{n} f_i(x). \tag{1}$$

With problem (1) a graph $G = (N, E)$ can be associated, where $N = \{1, ..., n\}$ is the set of nodes, and E is the set of edges $\{i, j\}$, i.e., pairs of nodes i and j that can directly communicate.

As it will be seen, graph G represents a collection of realizable communication links; actual algorithms that are considered here may utilize subsets of these links over iterations in possibly unidirectional, sparsified communications.

The assumption is that G is connected, undirected and simple (no self nor multiple links). Denote by Ω_i the neighborhood set of node i and associate an $n \times n$ symmetric, doubly stochastic matrix W with graph G. The matrix W has positive diagonal entries and respects the sparsity pattern of graph G, i.e., for $i \neq j$, $W_{ij} = 0$ if and only if $\{i, j\} \notin E$. On the other hand, it is important to note, that in the cases of unidirectional communication between the nodes, the graph instantiations over iterations (subgraphs of G) can be directed. Also, assume that $W_{ii} > 0, \forall i$.

It can be shown that $\lambda_1(W) = 1$, and $\bar{\lambda}_2(W) < 1$, where $\lambda_1(W)$ is the largest eigenvalue of W, and $\bar{\lambda}_2(W)$ is the modulus of the eigenvalue of W that is second largest in modulus. Denote by $\lambda_n(W)$ the smallest eigenvalue of W. There also holds $|\lambda_n(W)| < 1$.

The following optimization problem can be associated with (1),

$$\min_{x \in R^{ns}} \Psi(x) := \sum_{i=1}^{n} f_i(x_i) + \frac{1}{2\alpha} \sum_{i<j} W_{ij} ||x_i - x_j||^2, \tag{2}$$

where $x = \left(x_1^T, ..., x_n^T\right)^T \in R^{ns}$ is the optimization variable partitioned into $s \times 1$ blocks $x_1, ..., x_n$. The reasoning behind this transformation is the following. Assume that $s = 1$ for simplicity. Under the stated assumptions on matrix W, it can be shown that $Wx = x$ if and only if $x_1 = x_2 = ... = x_n$, so the problem (1) is equivalent to

$$\min_{x \in R^{ns}} F(x), \quad \text{s.t. } (I - W)x = 0, \tag{3}$$

where $F(x) := \sum_{i=1}^{n} f_i(x_i)$ and I is the identity matrix. Moreover, $I - W$ is positive semidefinite, so $(I - W)x = 0$ is equivalent to $(I - W)^{1/2}x = 0$. Therefore, (3) can be replaced by

$$\min_{x \in R^{ns}} F(x), \quad \text{s.t. } (I - W)^{1/2}x = 0, \tag{4}$$

In other words, the constraint $Wx = x$ enforces that all the feasible x_i's in optimization problem (3) are mutually equal, thus ensuring the equivalence of (1) and (3) and the equivalence of (1) and (4). Further, a penalty reformulation of (4) can be stated as

$$\min_{x\in R^{ns}} F(x) + \frac{1}{2\alpha} x^T (I - W)x, \tag{5}$$

where $\frac{1}{\alpha}$ is the penalty parameter. Therefore (5) represents a quadratic penalty reformulation of the original problem (1). After standard manipulations with the penalty part we obtain

$$\min_{x\in R^{ns}} F(x) + \frac{1}{2\alpha} \sum_{i<j} W_{ij} \left(x_i - x_j\right)^2, \tag{6}$$

which is the same as (2) for $s = 1$. These considerations are easily generalized for $s > 1$.

It is well known, [1], that the solutions of (1) and (2) are mutually close. More specifically, for each $i = 1, ..., n$, $||x_i^\circ - x^*|| = O(\alpha)$ where x^* is the solution to (1), $x^\bullet = \left(\left(x_1^\circ\right)^T, ..., \left(x_n^\circ\right)^T\right)^T$ is the solution to (2). In more details, Theorem 4 in [29] says that under strongly convex local costs f_i's with Lipschitz continuous gradients (see ahead Assumption 2.1 for details), the following holds, for all $i = 1, ..., n$:

$$\begin{aligned} \|x_i^\circ - x^\star\| &\leq \left(\frac{\alpha LD}{1 - \bar{\lambda}_2(W)}\right)\sqrt{4/c^2 - 2\alpha/c} + \frac{\alpha D}{1 - \bar{\lambda}_2(W)} \\ &= O\left(\frac{\alpha}{1 - \bar{\lambda}_2(W)}\right), \end{aligned} \tag{7}$$

$$D = \sqrt{2L\left(\sum_{i=1}^n f_i(0) - \sum_{i=1}^n f_i\left(x_i'\right)\right)}; c = \frac{\mu L}{\mu + L}. \tag{8}$$

Here, x_i' is the minimizer of f_i, L is the Lipschitz constant of the gradients of the f_i's, and μ is the strong convexity constant of the f_i's.

The usefulness of formulation (2) is that it offers a solution that is close (on the order $O(\alpha)$) to the desired solution of (1), while, unlike formulation (1), it is readily amenable for distributed implementation. A key insight known in the literature (see, e.g. [4, 30]) is that applying a conventional (centralized) gradient descent method on (2) precisely recovers the distributed gradient method proposed in [1]. In other words, it has been shown that the distributed method in [1] – that approximately solves (1) – actually converges to the solution of (2). This insight has been significantly explored in the literature to derive several distributed methods, e.g., [4, 5, 16]. The class of methods considered in this paper also exploits this insight and therefore harnesses formulation (2) to carry out convergence analysis of the considered methods.

2.2 Algorithmic framework

The algorithmic framework is presented in this Section. The framework subsumes several existing methods [12–17], and it also includes a new method that will be analysed in this paper.

Within the considered framework, each node i in the network maintains $x_i^k \in R^s$, its approximate solution to (1), where k is the iteration counter. In addition, let us associate a Bernoulli random variable z_i^k to each node i, that governs its communication activity at iteration k. If $z_i^k = 1$, node i communicates; if $z_i^k = 0$, node i does not exchange messages

with neighbors. When $z_i^k = 1$, node i transmits x_i^k to all its neighbours $j \in \Omega_i$, and it receives x_j^k, from all its active (transmitting) neighbours.

The intuition behind the introduction of quantities z_i^k is the following. It has been demonstrated (see, e.g., [12]) that distributed methods to solve (1) and (2) exhibit certain "redundancy" in terms of the utilized communications. In other words, it is not necessary to activate all communication channels at all times for the algorithm to be convergent. Moreover, communication sparsification may lead to convergence speed improvements in terms of communication cost [12]. Communication sparsification and introduction of the z_i^k's leads to less expensive but inexact algorithmic updates. A proper design of the z_i^k's can lead to a positive resolution of the inexact-less expensive updates tradeoff; see, e.g., [12] for details.

Assume that the random variables z_i^k are independent both across nodes and across iterations. Denote by $p_k = Prob\left(z_i^k = 1\right)$, assumed equal across all nodes. The quantity p_k is a design parameter of the method; strategies for setting p_k are discussed further ahead. Intuitively, a large p_k corresponds to "less inexact" updates but also to lower communication savings. With the considered algorithmic framework, solution estimate update at node i is as follows:

$$d_i^k = -\left[\left(M_i^k\right)^{-1}\left[\alpha\nabla f_i\left(x_i^k\right) + \sum_{j\in\Omega_i} W_{ij}\left(x_i^k - x_j^k\right)\xi_{i,j}^k\right]\right], \tag{9}$$

$$x_i^{k+1} = x_i^k + d_i^k. \tag{10}$$

Here, α is a positive parameter, known as the step-size. The values of α differ depending on the input data (See ahead Section 2.5). Further, $\xi_{i,j}^k$ is in general a function of z_i^k and z_j^k that encodes communication sparsification; and M_i^k is a local second order information-capturing matrix, i.e., the Hessian approximation.

The following choices of the quantities $\xi_{i,j}^k$ and M_i^k will be considered: 1) $\xi_{i,j}^k = 1$: no communication sparsification; 2) $\xi_{i,j}^k = z_i^k \cdot z_j^k$ bidirectional communication sparsification (that is, node i includes node j's solution estimate in its update only if both i and j are active in terms of communications); and 3) $\xi_{i,j}^k = z_j^k$ (unidirectional communication); that is, node i includes node j's solution estimate in its update whenever node j transmits, irrespective of node i being transmission-active or not.

Regarding the matrix M_i^k, two options can be considered. First, $M_i^k = I$ and this corresponds to first order methods, where one has no second order information included. Second option is $M_i^k = D_i^k$, where:

$$D_i^k = \alpha\nabla^2 f_i\left(x_i^k\right) + (1 - W_{ii})\, I. \tag{11}$$

This corresponds to the second order methods of DQN-type [16] (See ahead Section 2.6).

We now provide intuition behind the generic method (9)-(10) and the choices of $\xi_{i,j}^k$'s and M_i^k's. The method (9)-(10) corresponds to an inexact first order or an inexact second order method to solve (2) – and hence to approximately solve (1). The main source of inexactness is due to the sparsification ($\xi_{i,j}^k$'s). The bidirectional communication $\left(\xi_{i,j}^k = z_i^k \cdot z_j^k\right)$ is appealing as it preserves symmetry in the underlying weight matrix, which is known to be a beneficial theoretical property. On the other hand, the bidirectional sparsification is also wasteful in that a node ignores the received message from a neighbor if its own transmission to the same neighbor is not successful (see formula (9)). With respect to the

choice first versus second order method (the choice of M_i^k), the second order choice is computationally more expensive per iteration due to the Hessian computations; on the other hand, it can improve convergence speed iteration-wise.

The pseudocode for the general algorithmic framework is in Algorithm 1. A summary of all the considered methods within the framework of Algorithm 1 is given in Table 1.

Algorithm 1 Pseudocode for the proposed algorithmic framework

Require: at each node i: $\alpha > 0$; $\{W_{ij}\}_{j\in\Omega_i}$; $\{p_k\}_{k\geq 0}$

repeat

Each node i generates z_i^k and computes:

M_i^k and $\xi_{i,j}^k$, $j \in \Omega_i$

if $\xi_{i,j}^k = 1$ **then**

Each node i receives x_j^k from node $j, j \in \Omega_i$

end if

Each node i updates x_i^k via (9) – (10)

until a stopping criterion is met

2.3 Convergence analysis

In this section, a convergence analysis of the algorithm variant with unidirectional communications is carried out (See ahead *Method FUI* in Section 2.6). More precisely, in this section we assume the following choice of M_i^k and ξ_{ij}^k:

$$M_i^k = I, \quad \xi_{ij}^k = z_j^k. \tag{12}$$

To the best of our knowledge, except for a different estimation setting [14], this algorithm has not been studied before. The following assumptions are needed.

Assumption 2.1. *(a) Each function $f_i : \mathbb{R}^s \to \mathbb{R}$, $i = 1, ..., n$ is twice differentiable, strongly convex with strong convexity modulus $\mu > 0$, and it has Lipschitz continuous gradient with the constant L, $L \geq \mu$.*

(b) The graph G is undirected, connected and simple.

(c) The step size α in (2) satisfies $\alpha < \min\left\{\frac{1}{2L}, \frac{1+\lambda_n(W)}{L}\right\}$.

By Assumption 2.1, Ψ is strongly convex with modulus μ. Moreover, the gradient is Lipschitz continuous with the constant

$$L_\Psi := L + \frac{1 - \lambda_n(W)}{\alpha}. \tag{13}$$

Notice that Assumption 2.1 (c) implies that $\alpha < (1 + \lambda_n(W))/L$, which is equivalent to

$$\alpha < \frac{2}{L_\Psi}. \tag{14}$$

Let $x^k = \left(\left(x_1^k\right)^T, ..., \left(x_n^k\right)^T\right)^T$. We have the following convergence result for the first order method with unidirectional communications.

Theorem 2.1. *Let $\{x^k\}$ be a sequence generated by Algorithm 1, method FUI, and Assumption 2.1 holds. Then, the following results hold:*

(a) Assume that the sequence $\{p_k\}$ *converges to one as* $k \to \infty$. *Then, the sequence of iterates* $\left\{x^k\right\}$ *converges to* $x^\bullet$ *in the expected error norm, i.e., there holds:*

$$\lim_{k\to\infty} E\left[\left\|x^k - x^\bullet\right\|\right] = 0. \tag{15}$$

(b) Assume that the sequence $\{p_k\}$ *converges to one geometrically as* $k \to \infty$, *i.e.,* $p_k = 1 - \delta^{k+1}$, *for all k, Then, there holds:*

$$E\left[\left\|x^k - x^\bullet\right\|\right] = O\left(\gamma^k\right), \tag{16}$$

where $\gamma < 1$ *is a positive constant.*

(c) Assume that $p_k \geq p_{min}$ *for all k and for some* $p_{min} \in (0,1)$ *and that the iterative sequence* $\left\{x^k\right\}$ *is uniformly bounded, i.e., there exists a constant* $C_1 > 0$ *such that* $E\left[\left\|x^k\right\|\right] \leq C_1$, *for all k. Then, there holds:*

$$E\left[\left\|x^k - x^\bullet\right\|\right] \leq \theta^k \|x^0 - x^\bullet\| + (1 - p_{min})^2\, C_2, \tag{17}$$

where $C_2 = \frac{2nC_1}{\alpha\mu}$ *and* $\theta \in (0,1)$.

Theorem 2.1 demonstrates that the Algorithm 1 with sparsified and unidirectional communications converges. More precisely, as long as the sequence p_k converges to one, even arbitrarily slowly, the sequence $\left\{x^k\right\}$ converges to the solution of (2) in the expected error norm sense. When the convergence of p_k to one is geometric, we have that x^k converges geometrically, i.e., at a linear rate. Finally, when p_k stays bounded away from one, under the additional assumption that the sequence $\left\{x^k\right\}$ is uniformly bounded, the algorithm converges to a neighbourhood of the solution to (2), where the neighbourhood size is controlled by parameter p_{min} (the closer p_{min} to one, the smaller the error). This complements the existing results in [16] which concerns bidirectional communications.

Next, the proof of Theorem 2.1 will be carried out. To avoid notation clutter, let the dimension of the original problem (1) be $s = 1$. The proof relies on the fact that the method can be written as an inexact gradient method for minimization of Ψ. More specifically, it can be shown that the algorithm determined by (9) – (12) is equivalent to the following:

$$x^{k+1} = x^k - \alpha\left[\nabla\Psi\left(x^k\right) + e^k\right], \tag{18}$$

where $e^k = \left(e_1^k, ..., e_n^k\right)^T$ is given by

$$e_i^k = \frac{1}{\alpha}\sum_{j\in\Omega_i} W_{ij}\left(z_j^k - 1\right)\left(x_i^k - x_j^k\right) \tag{19}$$

and $e^k \in R^n$. Indeed, in view of (12), method (9)-(10) can be represented as

$$x^{k+1} = x^k - \alpha\nabla F\left(x^k\right) - \left(I - W_k\right)x^k, \tag{20}$$

where

$$F : \mathbb{R}^n \to \mathbb{R}, F(x) = \sum_{i=1}^{n} f_i\left(x_i\right), \tag{21}$$

$$\left[W_k\right]_{ij} = \begin{cases} W_{ij}z_j^k, & \text{if}\{i,j\} \in E, i \neq j, \\ 0, & \text{if}\{i,j\} \notin E, i \neq j, \\ 1 - \sum_{l\neq i}\left[W_k\right]_{il}, & \text{if} i = j. \end{cases} \tag{22}$$

Thus,

$$\begin{aligned} x^{k+1} &= x^k - \alpha\left(\nabla F\left(x^k\right) + \frac{1}{\alpha}\left(I - W_k\right)x^k \pm \frac{1}{\alpha}(I - W)x^k\right) \\ &= x^k - \alpha\left(\nabla\Psi\left(x^k\right) + \frac{1}{\alpha}\left(\left(I - W_k\right)x^k - (I - W)x^k\right)\right). \end{aligned} \tag{23}$$

Therefore, for each component i, the error is determined by

$$e_i^k = \frac{1}{\alpha}\left(\sum_{j\in\Omega_i} W_{ij} z_j^k\left(x_i^k - x_j^k\right) - \sum_{j\in\Omega_i} W_{ij}\left(x_i^k - x_j^k\right)\right), \tag{24}$$

and (19) follows.

Next we state and prove an important result. Here and throughout the paper, $||\cdot||$ denotes the vector 2-norm and the corresponding matrix norm.

Lemma 2.2. *Suppose that Assumption 2.1 holds. Then for each k we have*

$$||x^k - x^\bullet|| \le \theta^k||x^0 - x^\bullet|| + \alpha\sum_{t=1}^{k}\theta^{k-t}||e^{t-1}||, \tag{25}$$

where x^0 is the initial iterate and $\theta = \max\{1 - \alpha\mu, \alpha L_\Psi - 1\} < 1$.

Proof. Using (18) and the fact that $\nabla\Psi(x^\bullet) = 0$ we obtain

$$x^{k+1} - x^\bullet = x^k - x^\bullet - \alpha e^k - \alpha\left(\nabla\Psi\left(x^k\right) - \nabla\Psi\left(x^\bullet\right)\right). \tag{26}$$

Further, there exists a symmetric positive definite matrix B_k such that

$$\nabla\Psi\left(x^k\right) - \nabla\Psi\left(x^\bullet\right) = B_k\left(x^k - x^\bullet\right) \tag{27}$$

and its spectrum belongs to $[\mu, L_\Psi]$. Thus, we obtain

$$\|I - \alpha B_k\| \le \max\{1 - \alpha\mu, \alpha L_\Psi - 1\} := \theta. \tag{28}$$

Notice that the Assumption 2.1 (c) implies that $\theta < 1$ since (14) holds and $L \ge \mu$. Moreover, putting together (26) - (28), we obtain

$$\|x^{k+1} - x^\bullet\| \le \theta\|x^k - x^\bullet\| + \alpha\|e^k\| \tag{29}$$

and applying the induction argument we obtain the desired result. □

To complete the proof of parts (a) and (b) of Theorem 2.1, we need to derive an upper bound for $||e^k||$ in the expected-norm sense. In order to do so, it is needed to establish the boundedness of iterates x^k in the expected norm sense.

Lemma 2.3. *Let Assumption 2.1 hold, and consider the setting of Theorem 2.1 (a). Then, there holds $E\left[||x^k||\right] \le C_x$ for all k, where C_x is a positive constant.*

Proof. The update rule (20) can be written equivalently as follows

$$x^{k+1} = W_k x^k - \alpha\nabla F\left(x^k\right). \tag{30}$$

Introduce $\widetilde{W}_k = W_k - W$, and rewrite (30) as

$$x^{k+1} = Wx^k - \alpha\nabla F\left(x^k\right) + \widetilde{W}_k x^k. \tag{31}$$

Denote by x' the minimizer of F. Then, by the Mean Value Theorem, there holds

$$\begin{aligned}\nabla F\left(x^k\right) - \nabla F\left(x'\right) &= \underbrace{\left[\int_0^1 \nabla^2 F\left(x' + t\left(x^k - x'\right)\right) dt\right]}_{H_k}\left(x^k - x'\right) \\ &= H_k\left(x^k - x'\right) = H_k x^k - H_k x',\end{aligned} \tag{32}$$

and

$$x^{k+1} = \left(W - \alpha H_k\right) x^k + \widetilde{W}_k x^k + \alpha H_k x' - \alpha \nabla F(x'). \tag{33}$$

Note that $||H_k|| \leq L$, by Assumption 2.1. Also, note that $||W - \alpha H_k|| \leq 1 - \alpha\mu$, for $\alpha \leq \frac{1}{2L}$. Therefore, the following can be stated

$$\begin{aligned}||x^{k+1}|| &\leq (1 - \alpha\mu)||x^k|| + \underbrace{\alpha\left(L||x'|| + ||\nabla F\left(x'\right)||\right)}_{C'} \\ &\quad + ||\widetilde{W}_k|| \cdot ||x_k|| \\ &= (1 - \alpha\mu)||x^k|| + C' + ||\widetilde{W}_k|| \cdot ||x_k||.\end{aligned} \tag{34}$$

Next, $||\widetilde{W}_k||$ will be upper bounded. Note that

$$||\widetilde{W}_k|| \leq \sqrt{n}||\widetilde{W}_k||_1 \leq \sqrt{n}\sum_{i=1}^{n}\sum_{j=1}^{n}|\left[\widetilde{W}_k\right]_{ij}|. \tag{35}$$

Therefore,

$$||\widetilde{W}_k|| \leq 2\sqrt{n}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}\left(1 - z_j^k\right). \tag{36}$$

Taking expectation and using the fact that $E\left[z_j^k\right] = p_k$, for all k, it can be concluded that

$$E\left[||\widetilde{W}_k||\right] \leq \widetilde{C}\left(1 - p_k\right) \tag{37}$$

for some positive constant $\widetilde{C}$. Now, using independence of $\widetilde{W}_k$ and x_k, the following can be obtained from (34),

$$\begin{aligned}E\left[||x^{k+1}||\right] &\leq (1 - \alpha\mu)E\left[||x^k||\right] + C' + \left(1 - p_{k+1}\right)\widetilde{C}E\left[||x^k||\right] \\ &= \left(1 - \alpha\mu + \widetilde{C}\left(1 - p_{k+1}\right)\right)E\left[||x^k||\right] + C'.\end{aligned} \tag{38}$$

As $p_k \to 1$, i.e., $(1 - p_k) \to 0$, it is clear that, for sufficiently large k, there holds

$$E\left[||x^{k+1}||\right] \leq \left(1 - \frac{1}{2}\alpha\mu\right)E\left[||x^k||\right] + C'. \tag{39}$$

This implies that there exists a constant C_x such that $E\left[||x^k||\right] \leq C_x$, for all $k = 0, 1, \ldots$. □

Applying Lemma 2.3, the following result is obtained.

Lemma 2.4. *Suppose that the Assumption 2.1 holds and* $E\left(\left\|x^k\right\|\right) \leq C_1$ *for all* k *and some constant* C_1. *Then the error sequence* $\{\|e^k\|\}$ *satisfies*

$$E\left[||e^k||\right] \leq \left(1 - p_k\right) C_e, \tag{40}$$

for the constant $C_e = \frac{2n}{\alpha}\left(1 - p_{min}\right) C_1$.

Proof. The proof follows straightforwardly from (19) and Lemma 2.3. Consider (24). Then, $|e_i^k|$ can be upper bounded as follows:

$$|e_i^k| \leq \frac{1}{\alpha} \sum_{j \in \Omega_i} w_{ij} |1 - z_j^k| 2 \left\| x^k \right\|. \tag{41}$$

This yields:

$$\left\| e^k \right\| \leq \left\| e^k \right\|_1 = \sum_{i=1}^{n} \frac{2}{\alpha} \sum_{j \in \Omega_i} w_{ij} |1 - z_j^k| \left\| x^k \right\|. \tag{42}$$

Taking expectation while using independence of z_j^k and x^k, and using $E\left(\left\|x^k\right\|\right) \leq C_1$; $\sum_{j \in \Omega_i} \leq 1$; and $E\left(|1 - z_j^k|\right) = 1 - p_k$, the result follows. □

Now, Theorem 2.1 can be proved as follows.

Proof of Theorem 2.1. We first prove part (a). Taking expectation in Lemma 2.2, and using Lemma 2.4, we get

$$\begin{aligned} E\left[||x^k - x^\bullet||\right] &\leq \theta^k ||x^0 - x^\bullet|| + \alpha \sum_{t=1}^{k} \theta^{k-t} E\left[||e^{t-1}||\right] \\ &\leq \theta^k ||x^0 - x^\bullet|| + \alpha \sum_{t=1}^{k} \theta^{k-t} \cdot C_e \left(1 - p_{t-1}\right). \end{aligned} \tag{43}$$

Next, applying Lemma 3.1 in [31], it follows that

$$E\left[\left\|x^k - x^\bullet\right\|\right] \to 0, \tag{44}$$

as we wanted to prove.

Let us now consider the part (b). Note that, in this case, we have that $1 - p_k = \delta^{k+1}$, for all k. Specializing the bound in (43) to this choice of p_k, the following holds

$$E\left[||x^k - x^\bullet||\right] \leq \theta^k ||x^0 - x^\bullet|| + \alpha C_e \sum_{t=1}^{k} \theta^{k-t} \delta^t, \tag{45}$$

and using the fact that $s_k := \sum_{t=1}^{k} \theta^{k-t} \delta^t$ converges to zero R-linearly (see Lemma II.1 from [16]), we obtain the result.

Finally, we prove part (c). Here, we upper bound the term $(1 - p_{t-1})$ in (43) with $(1 - p_{min})$. For this case we obtain

$$\begin{aligned} E\left[||x^k - x^\bullet||\right] &\leq \theta^k ||x^0 - x^\bullet|| \\ &\quad + (1 - p_{min}) C_e \frac{1}{\mu}, \end{aligned} \tag{46}$$

which completes the proof of part (c).□

2.4 Implementation and infrastructure

A parallel implementation of Algorithm 1 was developed, using MPI [19]. The testing was performed on the AXIOM computing facility consisting of 16 nodes (8 x Intel i7 5820k 3.3GHz and 8 x Intel i7 8700 3.2GHz CPU - 96 cores and 16GB DDR4 RAM/node) interconnected by a 10 Gbps network.

Network configurations of grid and regular graphs are taken into consideration for graph G. A set of tests is conducted for the same data set with the same number of nodes for both types of graphs - d-regular graphs and grid.

The input data for the algorithm are read from binary files by the master process. The master process then scatters the data to other processes in equal pieces. If the data size is not divisible by the number of processes, then the remaining data is assigned to the master process. Therefore, the data are in the memory during computation and there is no Input/Output (I/O) operation performed while executing the algorithm.

The communication between the nodes is realized by creating a set of communicators – one for each node. The i-th communicator contains the i-th node as the master, and the nodes that are its neighbors. When sparsifying the communication between the nodes, the communicators should be recreated across the iterations, in order to ensure that only active nodes can send their results, see [11]. When using bidirectional communications, an active node is being included into its own communicator and into the communicators of its active neighbours. An inactive node is not included in the communicators of its neighbors, and also does not need its own communicator at the current iteration. In the case of unidirectional communication, an inactive node is included in its own communicator, but not in the communicators of its neighbors.

The data distribution process does not consume a large amount of the execution time. For example, considering a data set that contains a matrix of 5000×6000 elements and a vector of 5000 elements, the initial setup, including reading and scattering the data, as well as the creation of the communicators, takes about 0.3s per process. When compared to the overall run-time of the tests it represents a relatively small percentage. Regarding the case with the lowest execution time this percentage is 5%. On the other hand it is only 0.0007%, in the case with the highest execution time.

Regarding the stopping criteria, we let the algorithms run until $||\nabla\Psi\left(x^k\right)|| \leq \epsilon$, where $\epsilon = 0.01$. Note that the gradient $\nabla\Psi\left(x^k\right)$ is not computable by any node in a distributed graph G in general. In our implementation $\nabla\Psi\left(x^k\right)$ is maintained by the master node. While not being a realistic stopping criterion in a fully distributed setting, it allows us to adequately compare different algorithmic strategies,

The implementation relies on efficient LAPACK [32] and BLAS [33] linear algebra operations, applied on the nodes, while performing local calculations.

2.5 Simulation setup

The tests were performed on two types of graphs: d-regular and grid graphs with different number of nodes. We constructed the d-regular graphs in the following way. For 8-regular graphs, for each number of nodes n, we construct an 8-regular graph starting from a ring graph with nodes $\{1, 2, ..., n\}$ and then adding to each node i the links to the nodes $i-4$, $i-3$, $i-2$, and $i+2$, $i+3$, and $i+4$, where the subtractions and additions here are modulo n. The same principle was also used for 4-regular and 16-regular graphs used in this paper.

The tests are performed for the logistic loss functions given by

$$f_i(x) = \sum_{j=1}^{J} \mathcal{J}_{logis}\left(b_{ij}\left(x_1^\top a_{ij} + x_0\right)\right) + \frac{\tau}{n}||x||^2. \tag{47}$$

Here, $x = \left(x_1^T, x_0\right) \in \mathbb{R}^{s-1} \times \mathbb{R}$ represents the optimization variable and τ is the penalty parameter. The input values are $a_i \in \mathbb{R}^{s-1}$ and $b_i \in \mathbb{R}$.

The testing is performed on different versions of Algorithm 1 with sparsified communication, for both bidirectional and unidirectional communication strategies (see ahead Table 1).

The input data are represented as an $r \times (s-1)$ sized matrix of features, and an r sized vector of labels. Both the matrix and the vector are then divided into n parts corresponding to the nodes as explained in the previous section. We then vary n (and the corresponding graph G) and investigate the performance of Algorithm 1.

The following data sets were used for testing.

- The Conll data set [34, 35], that concerns language-independent named entity recognition. It has $r = 220663$ and $s = 20$ as the input data sizes. This data set is only used for comparing the performance of the algorithm between regular and grid graphs.
- The Gisette data set [36–38], known as a handwritten digit recognition problem. Its input data sizes are $r = 6000$ and $s = 5001$. The data set is used for testing the different alternatives of the algorithm as well as for determining the most suitable value of d for d-regular graphs.
- The YearPredictionMSD train data set is used to predict the release year of a song from audio features [37, 39, 40]. Here r and s are $r = 463715$ and $s = 91$. The data set is also used for testing the different alternatives of the algorithm.
- The MNIST data set represents a database of handwritten digits [41, 42], with input data sizes $r = 60000$ and $s = 785$. This data set is also used for testing the different alternatives of the algorithm.
- The Relative location of CT slices on axial axis data set (referred to as CT data set further on), containing features extracted from CT images [37, 43, 44]. The data sizes are $r = 53500$ and $s = 386$. This data set is also used for testing the different alternatives of the algorithm.
- The p53 Mutants data set [37, 45–48] (referred to as p53 data set further on) is used for modelling mutant p53 transcriptional activity (active or inactive) based on data extracted from biophysical simulations. The data set sizes are $r = 31159$ and $s = 5410$. The data set is also used for testing the different alternatives of the algorithm.

The parameters for Algorithm 1 are set according to the experimentally obtained conclusions. The value α can be defined as $\alpha = \frac{1}{KL}$, where L is the Lipschitz gradient constant and $K \in [10, 100]$, as proposed in [5]. The value of α can be fine-tuned according to the data set used for the tests. Increasing this value can lead to faster convergence. However, if the value is too large, then the algorithm might converge to a coarse solution neighbourhood. The values of α used for the mentioned 5 data sets are obtained experimentally and are listed below:

- $\alpha = 0.0001$ for the Gisette data set;
- $\alpha = 0.001$ for the p53 data set;
- $\alpha = 0.1$ for the YearPredictionMSD, MNIST, Conll and CT data sets.

A larger value of $\alpha = 0.1$ can be applied in the cases of relatively small number of features, compared to the number of instances (i.e. rows of data). Here, in all the 4 cases for $\alpha = 0.1$, the number of features is smaller than 1000.

The probability of communication p_k is set as follows: $p_k = 1 - 0.5^k$, where k is the iteration counter, or as $p_k = (k+1)^{-1}$. In other words, we consider an increasing and a decreasing sequence for the p_k's. The decreasing sequence for the probability is of interest

for analysis, as it gradually reduces the communication time over the iterations. This might require more iterations as the communication links are sparser. The increasing sequence for the probability may, on the other hand require less iterations, but those iterations are becoming increasingly more time consuming as the number of communication lines increases. It is of interest to investigate both possibilities.

The local second order information-capturing matrix M_i^k can be included to the computation as $M_i^k = D_i^k$, where D_i^k is defined as in (11), or it can be replaced by an identity matrix $M_i^k = I$. Both possibilities are of interest for testing as it is of interest to establish empirically if the additional computation required to solve the system of linear equations in (9) pays off. With $M_i^k = I$ we are performing (probably larger) number of cheaper iterations.

2.6 Description of the methods

Table 1 lists the different methods as alternatives of Algorithm 1, considering the solution update, defined in (9), (10) and (11). The naming convention for the methods was already described in the introductory section (see page 2).

Method SBC represents the initial version of the algorithm, used as the benchmark here, where *Method FBC* is its first order equivalent. These methods does not utilize any communication sparsification.

Note that *Methods FBI, FBD, FUI, FUD, SBI, SBD, SUI, SUD* use sparsification with either increasing or decreasing communication probabilities p_k. The rationale for choosing a linearly increasing p_k and a sub-linearly decreasing p_k is adopted according to insights available in the literature; see, e.g., [16], [14]. While it is possible to consider other choices and fine-tuning of the sequence p_k, this topic is outside of the paper's scope. Our primary aim is to investigate the feasibility and performance of increasing and of decreasing sequence of p_k's relative to the always-communicating strategy (*Method SBC* and *Method FBC*), as well as relative to the unidirectional versus bidirectional communication, and the first order versus second order methods.

The convergence analysis for the novel method with unidirectional communication *Method FUI* is presented here, where *Methods SUI* and *SUD*, that also rely on unidirectional communication, remain open for theoretical analysis. The *Methods FBI, FBD, SBI* and *SBD*, using bidirectional communication are already analysed in the literature (see [12–17]).

The listed methods and data sets described before are used to derive some empirical conclusions. As expected, the analysis of obtained results provides some insights about the optimal number of nodes for different setups. Also, the advantages of particular methods are clearly visible and one can estimate the usefulness of sparsification based on these results, keeping in mind that the tests might be influenced by the selection of data sets. Nevertheless, we believe that the obtained insights are useful.

3 Results and Discussion

We now present the experimental results. First, we investigate the behaviour of the Algorithm 1 for two types of graphs - d-regular graphs and grid graphs. After that, we perform a sequence of tests using all the methods and the data sets stated above on d-regular graphs. These test are used to gain insight into effectiveness of different sparsification alternatives and differences between the first and second order methods in the framework of Algorithm 1.

3.1 Analysis of different graph types

The tests on different graph types are performed using the data sets Conll and CT with *Method SBC*. Figures 1 and 2 represent a performance comparison between the executions of the algorithm using different d-regular and grid graphs with *Method SBC* on CT and Conll data set, respectively.

Observing Fig. 1, it can be clearly concluded that d-regular graphs perform better than grid graphs, which becomes more evident with increasing number of nodes. However, d-regular graphs perform similarly on this data set for different values of d. The execution times for $d = 4$ and $d = 8$ are almost the same here. Therefore, it is important to examine the performance for different graphs on another data set. From Fig. 2, it is evident that the execution time decreases until the optimal number of nodes is reached, and starts to grow after that point. The same trend is present in Fig. 1, but the optimal number of nodes is higher here. Figure 2 clearly shows the difference between d-regular and grid graphs. It also identifies 8-regular graphs as the most suitable choice for different number of nodes. Therefore, in the rest of the paper we consider 8-regular graphs, based on the derived empirical conclusions. For the cases, where the number of nodes n is smaller than 8, the value $d = n - 1$ is used, leading to all-to-all graphs for $n < 8$.

3.2 Analysis of scaling properties

A sequence of tests with different number of computational nodes n is performed next to give an insight into the most suitable number of nodes for the data sets. Figures 3 and 4 represent examples of the scaling properties of the algorithm, for *Method FBI* on the YearPredictionMSD data set and for *Method FUI* on the MNIST data set, respectively. Here, when varying n we keep the graph structure to the 8-regular graph. The optimal number of nodes can be identified in both cases. These graphs obviously show the usual expected trend where the execution time decreases until the optimal number of nodes is reached, while after that further enhancement in number of nodes leads to time increase. Intuitively, the larger number of workers n means that the same overall workload is parallelized over more workers, leading to time reduction. However, the beneficial effect is lost for sufficiently large n when the communication overhead time starts to dominate. Interestingly, the optimal number of nodes is mostly constant for the first order methods as well as for the second order methods, irrespective of the data set.

Table 2 shows the percentages of successful tests for all methods, i.e., of tests that satisfy the stopping criteria $\|\Phi\left(x^k\right)\| < 0.01$ within the maximal execution time of 15 hours. In the failed tests the iterations are also approaching the solution, but they did not reach

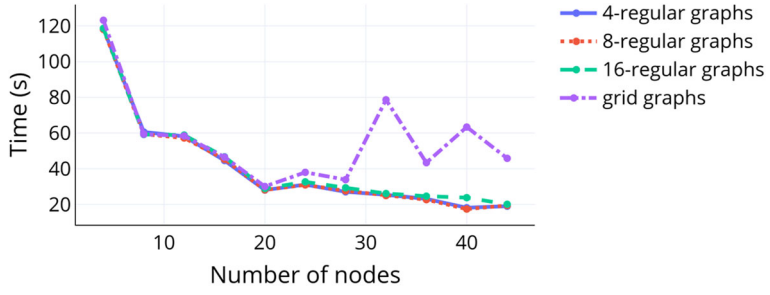

Fig. 1 Comparing the execution time using regular and grid graphs, CT data set, for *Method SBC*

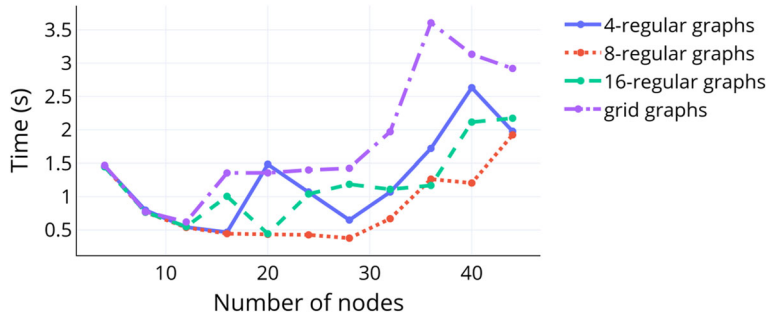

Fig. 2 Comparing the execution time using regular and grid graphs, Conll data set, for *Method SBC*

it within the given time limit. The results indicate that the first order methods are better choice in this environment as *Method SUD* is the one with the smallest number of successful tests. This fact can be easily explained as the method computes the expensive second order direction and the communication probability decreases while the communications are unidirectional. All this leads to the lack of communication epochs needed to ensure convergence during the time consuming iterations.

3.3 Analysis and comparison of execution time for the introduced methods

Table 3 lists the execution time for each of the 10 methods, for the p53 data set and 20 nodes. The maximal execution time, i.e. the time for the slowest process, is taken into account for all the cases. As this amount of time can vary on different processes, all processes are waiting for the slowest one in the communicator in order to successfully exchange the data. All first order methods introduce significant execution time reduction. In this case, *Method FBD* has the best performance. When comparing *Method FBC* to *Method SBC*, it is clear that the computation of second order direction d_i^k significantly increases the execution time. Reducing the amount of communication across the iterations with *Method FBD* leads to even faster execution here. However, this behaviour may be highly dependent on the nature of the data set. The algorithms for p53 data set converge fast, within relatively small number of iterations. An equally important aspect here is also the fact that *Method FUD*, using unidirectional communication and decreasing communication probability performs better than *Method FBI*, with bidirectional and

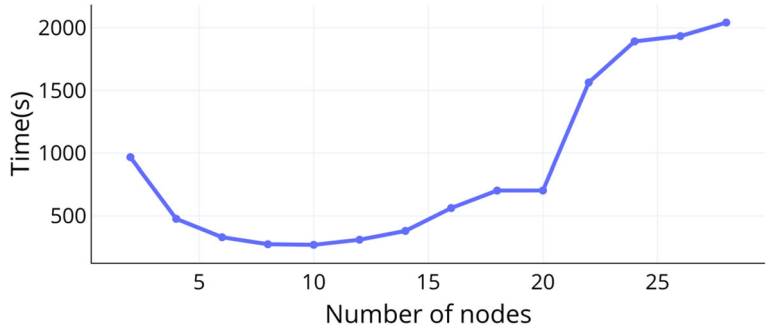

Fig. 3 Scaling properties of *Method FBI*, for the YearPredictionMSD data set

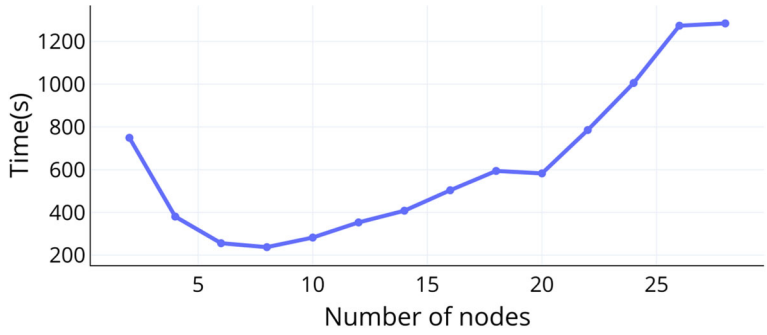

Fig. 4 Scaling properties of *Method FUI*, for the MNIST data set

increasing communication. Observing the execution times for the second order methods proves that introducing communication sparsification mostly does not pay off as the computation of the second order direction is time consuming.

As the nature of the data can highly influence the results, let us consider another example. Table 4 also contains the execution time for each of the 10 algorithms with 12 nodes for the MNIST data set (*Method SUD* does not converge for the given execution time limit). The behaviour of this data set differs from the p53 data set, observed in Table 3. For example, for 12 nodes *Method FBD* requires 4795 iterations to converge for the MNIST data set. When considering the p53 data set for the same setup with 12 nodes, it converges after only 3 iterations. However, the conclusions based on Table 4 are very similar to those from Table 3. In fact, it seems that the properties of particular methods are similar as long as the data sets are of similar volume.

Figures 5 and 6 represent the execution times for first order methods with communication sparsification, i.e. *Methods FBI, FBD, FUI, FUD* for the CT and Gisette data set, respectively. From Fig. 5, it can be concluded that the optimal number of nodes for *Methods FBI, FUI* and *FUD*, is the same value $n = 6$. However, *Method FBD* performs differently. It shows lower execution time values generally, and its optimal number of nodes is $n = 10$. Similar conclusions could be made based on Fig. 6. Here, the optimal number of nodes for *Methods FBI, FUI* and *FUD* is again the same, $n = 8$. *Method FBD* also performs differently here, with lower execution time values, compared to other first order methods. The optimal number of nodes for the second order methods tends to be a larger

Table 2 Percentages of successful test with respect to the overall number of tests

Method	Percentage
FBI	99.1
FBD	100
FUI	100
FUD	100
FBC	100
SBC	98.3
SBI	84.1
SBD	95.8
SUI	95.8
SUD	35

Table 3 Execution time for different variations of Algorithm 1, for 20 nodes, p53 data set

Method	Execution time (s)
FBI	4.64
FBD	1.89
FUI	6.04
FUD	3.56
FBC	3.16
SBC	9661.42
SBI	43126.71
SBD	22683.84
SUI	22029.20
SUD	9651.77

number. This is a direct consequence of the fact that the time consuming computations for the direction are faster with smaller portions of data on a node.

3.4 Analysis of the effects of communication sparsification

Figure 7 represents the average cost reduction for different number of nodes, compared to the method of the weakest performances for each data set, where the average is taken across different data sets. These tests were performed for first order methods with communication sparsification, i.e., *Methods FBI, FBD, FUI* and *FUD*. For each data set, we divide the execution time for a given number of nodes with the worst execution time on the same data set, and compute the average value over methods for all the data sets, for different numbers on nodes. The conclusions based on this figure are consistent with the ones in Figs. 5 and 6. *Method FBD* has the best performance properties. Also, for each method, an optimal number of nodes can be identified.

An evaluation of the algorithm execution with different sequences $\{p_k\}$ that stay bounded away from one as k grows large is presented in Fig. 8. The unidirectional, first order method was tested on the Conll data set, using $\alpha = 0.1$. We observed the value of Ψ as in (2) during the execution of the algorithm. The value of Ψ decreases over time for all choices of p_k, as expected. The zoomed part of the figure is included in order to present the last few seconds of the execution before reaching the minimal values of Ψ. Figure 8 shows that for different values of p_k the iterative sequences do not converge to the same value, but also that for the constant p_k choices the obtained limiting values are close.

Table 4 Execution time for different variations of Algorithm 1, for 12 nodes, MNIST data set

Method	Execution time (s)
FBI	336.31
FBD	118.16
FUI	353.31
FUD	342.59
FBC	161.33
SBC	19045.00
SBI	3124.56
SBD	11853.99
SUI	12259.79
SUD	N/A

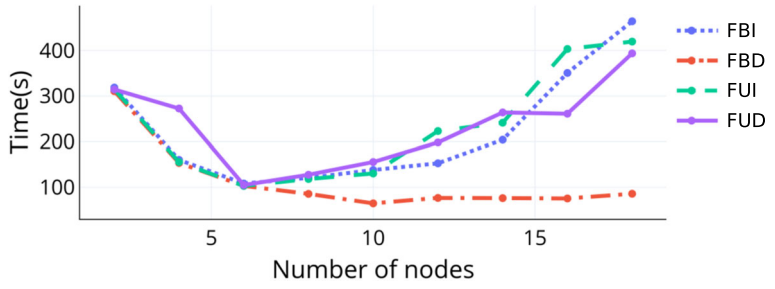

Fig. 5 Execution times for the first order *Methods FBI, FBD, FUI, FUD* on CT data set

Figures 9 - 16 displays the performance profile [49] for the described methods. Performance profiles enable evaluating the performance of different solvers running on a large number of tests. We consider the execution time as the comparison criterion. To compute the performance profile let us denote the execution time for a method M_i and test problem j by T_i^j. Then, given the value on the x-axis $\beta \geq 1$, the method M_i obtains a point for the performance on test j if there holds $T_i^j \leq \beta T_{min}^j$, where T_{min}^j is the smallest execution time of all tested methods considering that problem, i.e., $T_{min}^j = \min_i T_i^j$. The performance profile for a given β of the method M_i is then calculated as the number of points divided by the number of the performed tests. For example, on the y-axis where the parameter $\beta = 1$, we obtain the statistical probability that the method is the best one among all the tested methods in terms of the execution time. It is noticeable that the value range on the x axis is large, on these figures. This is due to the fact that there are very large differences in execution times, ranging from a few seconds to values larger that 18000 seconds.

Figure 9 shows the performance profile for all the tests on all data sets for the 10 methods, where Figs. 10 and 11 display the performance profile for first and second order methods, respectively. Figures 9 and 10 identify *Method FBD* as the best choice within the framework for Algorithm 1. Observing the methods without sparsification, i.e. *Methods SBC* and *FBC*, Fig. 9 indicates that the first order method, *Method FBC*, performs better than the second order method, *Method SBC*. The same is true if we consider the methods with sparsification. Considering methods with decreasing communication probability and using bidirectional communication, *Method FBD* performs clearly much better than *Method SBD*. When comparing the other first and second order methods using the same sparsification (*Method FBI* and *Method SBI*, *Method FUI* and *Method SUI*, *Method*

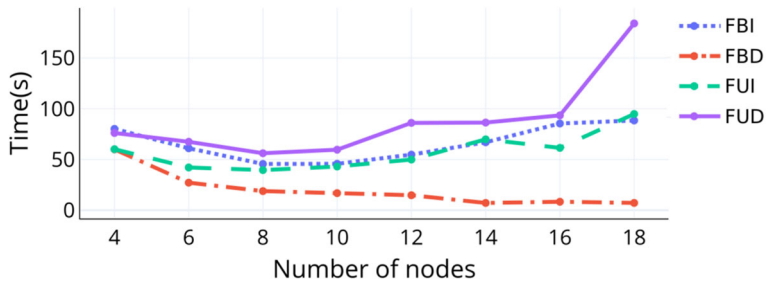

Fig. 6 Execution times for the first order *Methods FBI, FBD, FUI, FUD* on Gisette data set

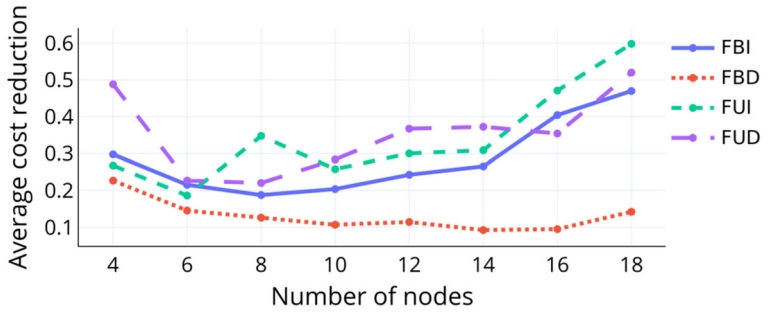

Fig. 7 Average cost reduction compared to the worst relevant tested method, *Methods FBI, FBD, FUI, FUD*

FUD and *Method SUD*), first order methods performs better in 61% of test cases. Also, the convergence rate for first order methods is higher (See Table 2). It can also be concluded that the sparsification of second order methods gives no advantages probably because the computation of the second order direction is time consuming. Furthermore, with communication sparsification the second order information is incorporated only partially and hence it does not provide enough advantage to compensate for computational load. On the other hand, communication sparsification can be beneficial for the first order methods, as evidenced by *Method FBD*. Generally, the best performing method is a first order method using the appropriate sparsification (bidirectional with decreasing communication probability), *Method FBD*.

Figure 12 represents the performance profile for the tests on the Gisette data set. Here, *Method FBD* can be also identified as the most suitable, followed by *Method FBC*, and later by *Method FUI*, *Method FBI* and *Method FUD*, where the second order methods show poorer performance profiles. The dimension s for this data set is a large value $s = 5001$, resulting with time consuming calculations in the second order methods as the Hessian approximation matrices are of large dimensions. Therefore, the first order methods perform better than second order methods. Figure 14 displays the performance profile for the tests on the p53 data set. The conclusions for this data set, are very similar to those for Fig. 12. Similarly, the dimension s is also a larger value here, $s = 5410$, so the first order methods also performs better than the second order methods and again, *Method FBD* represents the best choice. Similar conclusions are emerging from Fig. 13, that represents the performance profile for the MNIST data set. The dimension $s = 785$ is around 6 times smaller here, compared to Gisette and p53 data sets, but the dimension $r = 60000$

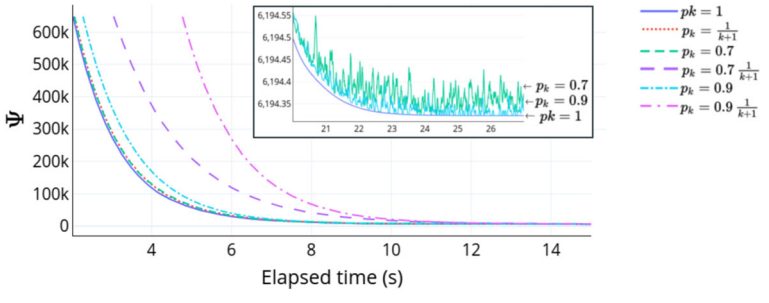

Fig. 8 Comparison between using different values of $p_k \leq 1$, first order method, unidirectional communication, Conll data set

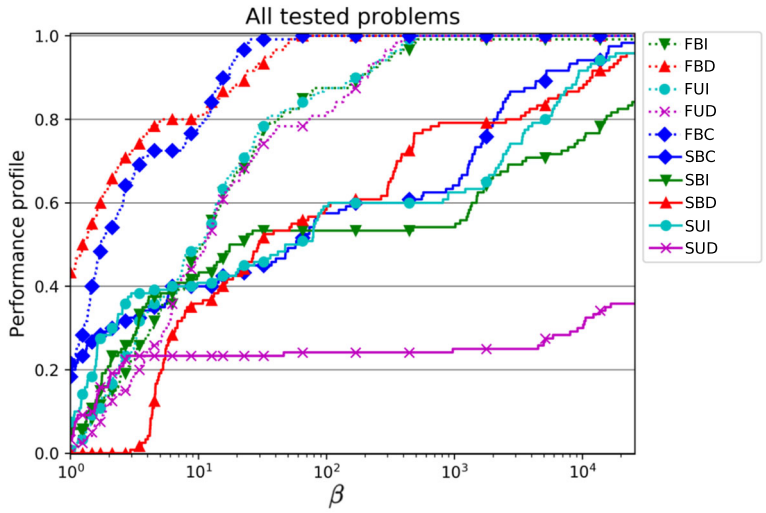

Fig. 9 Performance profile for the all 10 methods, based on all the performed tests

is 10 times larger than for Gisette, and 2 times larger than for p53. This results with similar load when distributing the data and calculation of the second order direction is too costly again.

The performance profile for the CT data set is displayed on Fig. 15. Here, the second order method *Method SUI* dominates, as the data set dimension $s = 386$ enables faster calculations of the second order direction. Comparison between the first and second order methods with the same communication sparsification yields the following conclusion - with the increasing communication probability the second order methods (*Methods SBI* and *SUI*) perform better (for both unidirectional and bidirectional communication). With the decreasing communication probability the first order methods (*Methods FBD* and *FUD*) give better results.

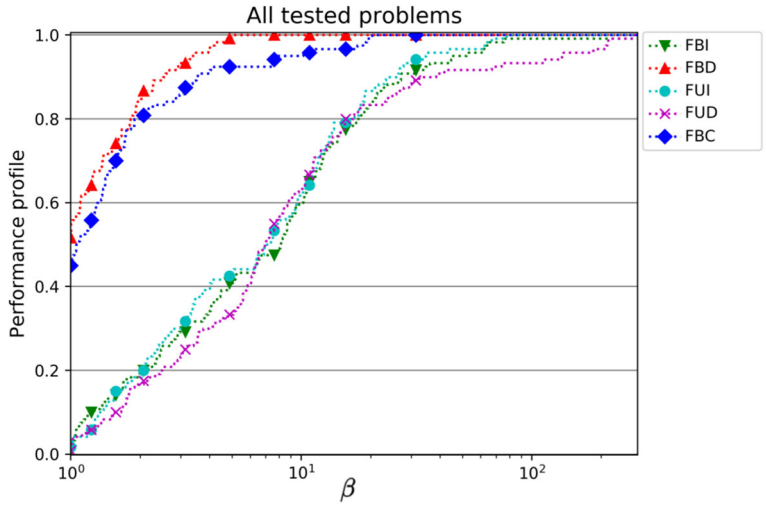

Fig. 10 Performance profile for first order *Methods FBI, FBD, FUI, FUD, FBC*, for all performed tests

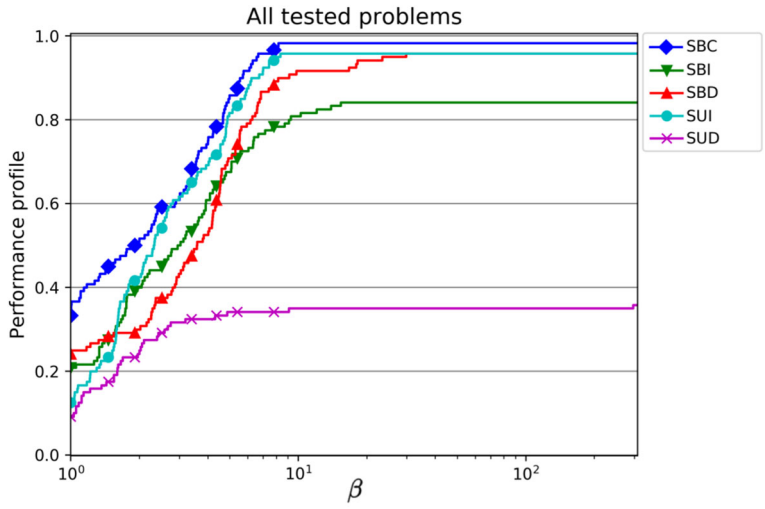

Fig. 11 Performance profile for second order *Methods SBC, SBI, SBD, SUI, SUD*, for all performed tests

Figure 16 represents the performance profile for the YearPredictionMSD data set. Here, the dimension $s = 91$ is the smallest among the observed data sets. Therefore the second order methods performs better. But the sparsification does not improve the first order nor the second order methods for these data. This fact might be explained by the large dimension $r = 463715$, and therefore each node gets a large subset. Sparsifying the communication means ignoring a large portion of data on idle nodes, even if there is only one idle node. Thus, the gradient and Hessian are poorly approximated with idling.

3.5 Comparison of Algorithm 1 to ADMM

As problem (1) can be solved using the Alternating Direction Method of Multipliers (ADMM) [11], we compared Algorithm 1 to an ADMM implementation for logistic regression [50], on the Conll data set. More precisely, the method in [11] solves problem

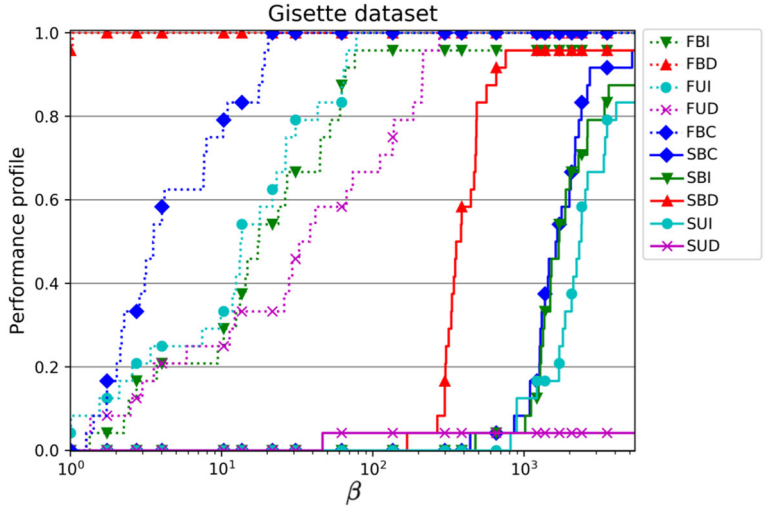

Fig. 12 Performance profile for all 10 methods, for the tests performed on the Gisette data set

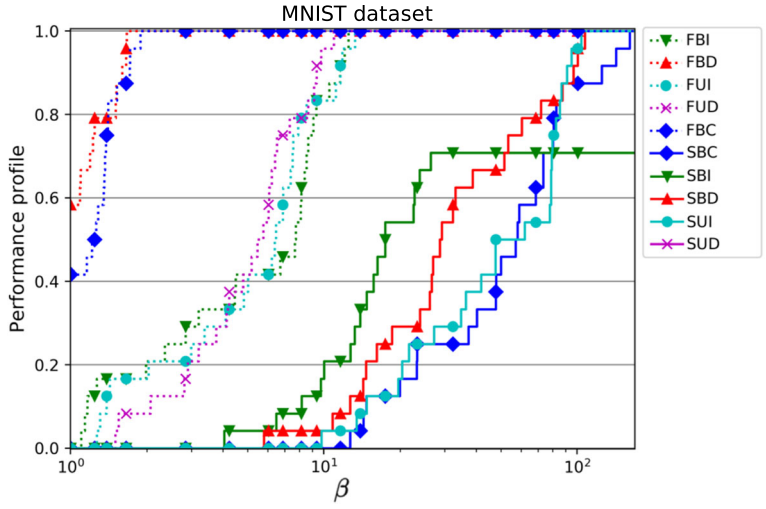

Fig. 13 Performance profile for all 10 methods, for the tests performed on the MNIST data set

(1) assuming the presence of a central node that communicates to all other nodes in the network. Henceforth, we adapt our algorithmic framework to the latter setting by letting the underlying network G to be fully connected and by setting the matrix W to have all its entries equal $1/n$. The comparison between the second order *Methods SBC* and *SBI* and ADMM is shown in Table 5. We calculate the value of $\Phi^k = \frac{1}{n}\sum_{i=1}^{n} f\left(x_i^k\right)$, i.e., the average global cost in (1) averaged across all nodes' estimates, at the end of each iteration and we also measure the execution time. The second column in Table 5 represents the time required to satisfy the condition $\frac{\Phi^k - f^*}{f^*} < 0.1$. Here, $f^\star$ is numerically evaluated by ADMM. The rationale for this comparison is the following. All the methods converge to a neighborhood of the solution to (1), while ADMM converges to the exact solution of (1). Therefore, it is meaningful to compare the times that each method needs to reach a certain accuracy level, measured with respect to the cost function in (1). We tested all the

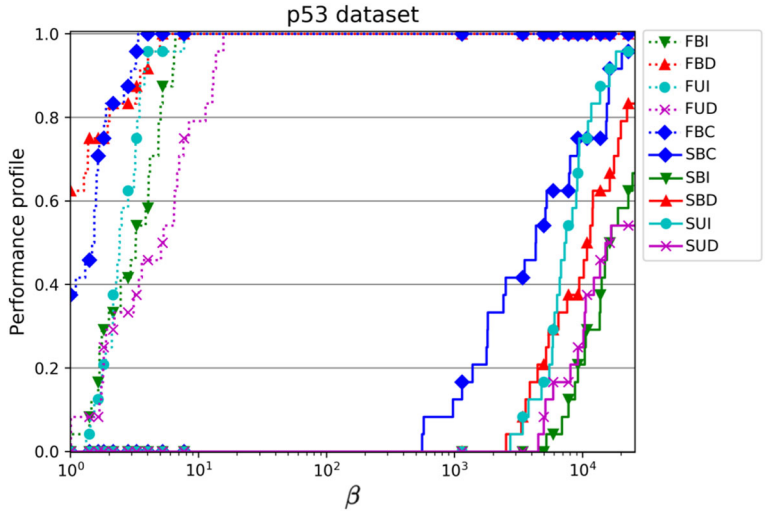

Fig. 14 Performance profile for all 10 methods, for the tests performed on the p53 data set

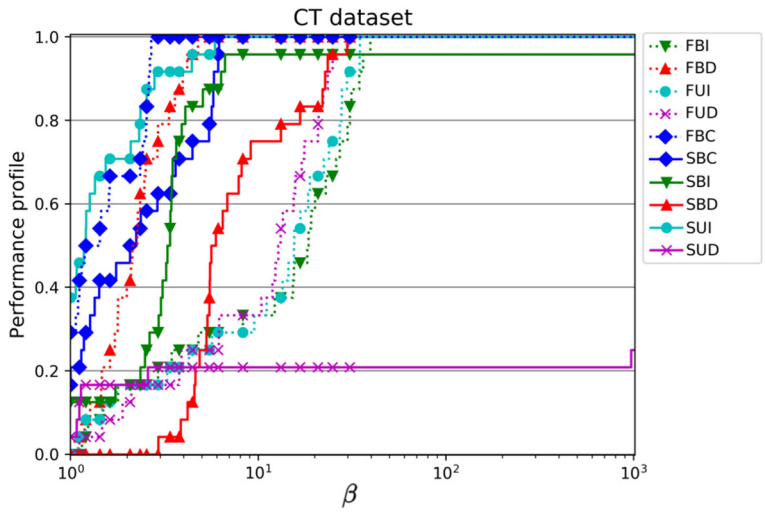

Fig. 15 Performance profile for all 10 methods, for the tests performed on the CT data set

methods and finally included the results for the best performing second order methods, i.e. *Methods SBC* and *SBI*. More precisely, *Method SBI* (a second order method with sparsification) is here the best performing method across all methods, while *Method SBC* is taken as the baseline (second order) method without sparsification. The fact that second order methods perform better than first order methods here is consistent with our previous conclusion that for smaller data sets, second order methods perform better than first order methods. It is clear that our second order methods converge faster than ADMM. Figure 17 shows the comparison between *Method SBI* and ADMM. *Method SBI* takes a larger number of significantly faster iterations, compared to ADMM, and hence results with shorter execution time needed to approach Φ^*.

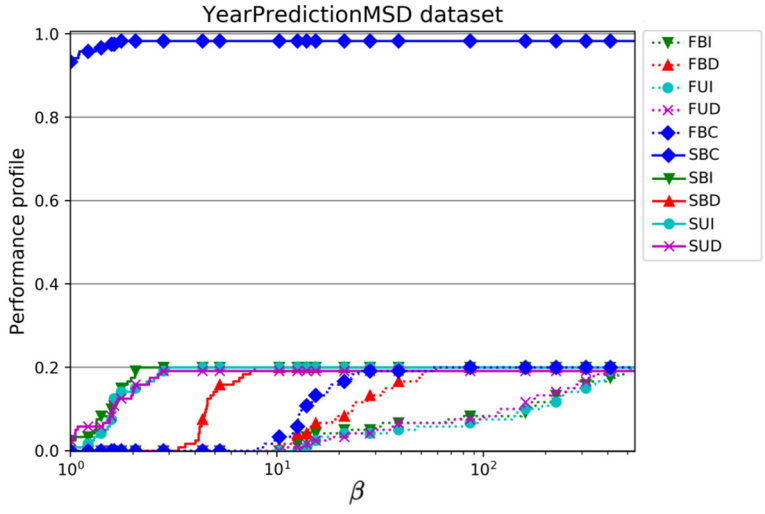

Fig. 16 Performance profile for all 10 methods, for the tests perfomed on the YearPredictionMSD data set

Table 5 Comparison of the second order *Methods SBC* and *SBI* with ADMM

Method	Execution time
ADMM	4.487
Method SBC	0.247
Method SBI	0.226

4 Conclusions

In this paper, we consider a class of first and second order distributed optimization methods which utilize different versions of the communication sparsification strategies. While the framework subsumes several existing recent methods, we also introduce a novel method with unidirectional communication and give its convergence analysis.

The paper provides a comprehensive empirical evaluation of various communication sparsification strategies on a HPC cluster. The tests of the algorithms without communication sparsification as well as with sparsified communications for different number of nodes [12, 16] are described in this paper. The overall execution time is measured for different data sets in order to identify the most suitable methods for different setups.

The tests were performed on an MPI cluster with a usual configuration, where each cluster node contains one processor with 6 CPU cores, and the nodes are connected by an Ethernet network with speed of 10Gbps. Therefore, we do not expect variations in the behavior of the tested programs on other MPI clusters. On the other hand, execution and results may depend on the speed of the cores themselves and on the speed of the network. Given that we used a cluster with eighth-generation Core i7 cores, a performance jump can be expected if newer-generation CPUs and / or more powerful Xeon processors were used. This effect would refer to the shortening of the absolute execution time per core, but overall performance characteristics would remain the same. The scaling properties would still be present, as well as the preferences of certain methods for the specified scenarios regarding the data. The factor that can mostly affect the execution of the program is the speed of the network. In the case of clusters with higher network speeds, the general expectation is to achieve good program performance with more nodes than in our experiments. In these cases, communication saturation, which we have shown to be present in this type of algorithm implementations, could only occur with more nodes involved than in our experiments (see Fig. 3 and 4, as well as Fig. 5 and 6 and the corresponding descriptions in Sections 3.2 and 3.3 respectively, that show these results for our experiments). In other words, increasing the network speed would be a crucial factor that would increase

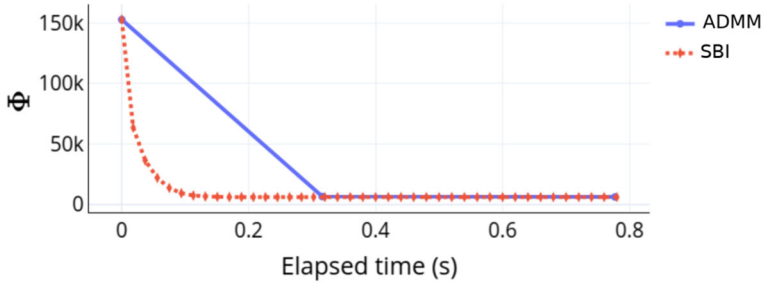

Fig. 17 The comparison between ADMM and Method SBI on Conll data set

the number of nodes on which the proposed implementations can be executed efficiently. This could result in different values for the optimal number of cores in different setups, compared to the results on Figs. 3–6.

The presented analysis also shows the expected scaling properties of the developed methods, starting from the differences in the optimal number of nodes for particular data set in consideration. The performance profile is used for the comparison between the proposed methods. It clearly identified that the first order methods perform much better with larger volumes of data, where for smaller data sets the second order methods are more suitable. For data sets with larger number of features (10^3 or more in our tests), the portions of data that the processes work on demand a significant amount of time to calculate the second order updates. If the number of samples is also larger (larger than 10^3 for our tests), it additionally burdens the execution. This is the reason why the first order methods perform better on larger data sets. The first order methods converge within a larger number of iterations, but those iterations are multiple times faster than for the second order methods. When the data set is smaller obtaining the second order information is not costly as the processes are working on small data portions. On these data sets the second order methods perform better as they converge within smaller number of iterations than the first order methods, while the second order iterations are negligibly slower than for the first order methods.

The method with bidirectional communication and decreasing communication probability (*Method FBD*) is identified as the best performing first order method. This method also shows the best performance globally, when observing all the tests on all 5 data sets. The fact that the bidirectional method performs better than the unidirectional method in most of the cases is a consequence of enabling exchange only between active nodes. Unidirectional methods require additional communication lines, in order to enable receiving data for idle nodes from their neighbors. The gain from solution update for the idle nodes can be slightly smaller than the cost of the communication to achieve that update. The decreasing probability enables more communication in the beginning of the execution. Later, the communication becomes sparse, but at the same time the solution becomes closer to the desired one, so that it does not require much communication any more. This is the reason why decreasing communication probability with a bidirectional method represents an optimal choice. However, the other methods with communication sparsification also showed satisfactory performance. The tests showed that, in general, communication sparsification can significantly improve performance. This serves as motivation for using communication sparsification in the described framework. It is also shown that communication sparsification does not introduce performance improvement with second order methods in general.

An important aspect of tests is the comparison between bidirectional and unidirectional communication. One conclusion is that unidirectional communication strategy works in the framework for Algorithm 1, and thus confirm the theoretical results. Besides that, this strategy yields lower execution time than the bidirectional communication strategy for some test cases. All these conclusions might be influenced by the considered data sets but nevertheless provide significant empirical evidence.

Further evaluation of unidirectional communication can be an interesting future task. Another challenging direction might be further implementation for very large data sets that cannot be held in memory.

Abbreviations
MPI: Message Passing Interface; HPC: High Performance Computing; DQN method: Distributed Quasi Newton method; I/O: Input/Output; ADMM: Alternating Direction Method of Multipliers

Acknowledgements
This work is supported by the I-BiDaaS project, funded by the European Commission under Grant Agreement No. 780787. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein. The authors gratefully acknowledge the AXIOM HPC facility at Faculty of Sciences, University of Novi Sad, where all the numerical simulations were run.

Authors' contributions
LF developed the implementation of the algorithm and performed the empirical evaluations. DJ, NK and NKJ contributed with the theoretical advances and design of methods. SS contributed to the experimentation and methods design equally. All authors participated in the main research flow development and in writing and revising the manuscript. All authors read and approved the final manuscript.

Authors' information
All authors are with Department of Mathematics and Informatics, Faculty of Sciences, University of NoviSad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia. e-mail: (lidija.fodor@dmi.uns.ac.rs; dusan.jakovetic@dmi.uns.ac.rs; natasa@dmi.uns.ac.rs; natasa.krklec@dmi.uns.ac.rs; srdjan.skrbic@dmi.uns.ac.rs).

Funding
Not applicable.

Availability of data and materials
The data sets used during the current study are available in:

- the UCI Machine Learning repository, [http://archive.ics.uci.edu/ml] [37] (Gisette [36, 38], YearPredictionMSD [39, 40], CT [43, 44] and p53 [45–48] data sets)
- the Language-Independent Named Entity Recognition II web site [https://www.clips.uantwerpen.be/conll2003/ner/] [34, 35] (the Conll data set)
- the MNIST DATABASE of Handwritten Digits web site [http://yann.lecun.com/exdb/mnist/] [41, 42] (the Mnist data set).

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 19 August 2020 Accepted: 17 May 2021
Published online: 01 June 2021

References

1. A. Nedic, A. Ozdaglar, Distributed subgradient methods for multi-agent optimization. IEEE Trans. Autom. Control. **54**(1), 48–61 (2009). https://doi.org/10.1109/tac.2008.2009515
2. S. S. Ram, A. Nedich, V. V. Veeravalli, Distributed stochastic subgradient projection algorithms for convex optimization. J. Optim. Theory Appl. **147**(3), 516–545 (2010). https://doi.org/10.1007/s10957-010-9737-7
3. D. Jakovetic, J. M. F. Xavier, J. M. F. Moura, Fast distributed gradient methods. IEEE Trans. Autom. Control. **59**(5), 1131–1146 (2014). https://doi.org/10.1109/tac.2014.2298712
4. A. Mokhtari, Q. Ling, A. Ribeiro, Network newton distributed optimization methods. IEEE Trans. Signal Process. **65**(1), 146–161 (2017). https://doi.org/10.1109/tsp.2016.2617829
5. D. Bajović, D. Jakovetić, N. Krejić, N. Krklec Jerinkić, Newton-like method with diagonal correction for distributed optimization. SIAM J. Optim. **27**(2), 1171–1203 (2017). https://doi.org/110.1137/15m1038049
6. A. Mokhtari, Q. Ling, A. Ribeiro, Network newton-part II: Convergence rate and implementation. arXiv: Optimization and Control (2015). arXiv preprint arXiv:1504.06020
7. K. Zhang, Z. Yang, H. Liu, T. Zhang, T. Basar, in *Proceedings of the 35th International Conference on Machine Learning, vol. 80*. ed. by J. Dy, A. Krause, Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents (PMLR, 2018), pp. 5872–5881
8. J. Shamma, *Cooperative Control of Distributed Multi-Agent Systems*. (Wiley-Interscience, USA, 2008). https://doi.org/10.1002/9780470724200
9. A. Salkham, R. Cunningham, A. Garg, V. Cahill, in *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, A collaborative reinforcement learning approach to urban traffic control optimization, vol. 2 (IEEE, Sydney, NSW, Australia, 2008), pp. 560–566. https://doi.org/10.1109/WIIAT.2008.88
10. R. Roche, B. Blunier, A. Miraoui, V. Hilaire, A. Koukam, in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, Multi-agent systems for grid energy management: A short review (IEEE, Glendale, 2010), pp. 3341–3346. https://doi.org/10.1109/IECON.2010.5675295
11. S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011). https://doi.org/10.1561/2200000016
12. D. Jakovetić, D. Bajović, N. Krejić, N. Krklec Jerinkić, Distributed gradient methods with variable number of working nodes. IEEE Trans. Signal Process. **64**(15), 4080–4095 (2016). https://doi.org/10.1109/TSP.2016.2560133

13. A. K. Sahu, D. Jakovetic, D. Bajovic, S. Kar, Communication-efficient distributed strongly convex stochastic optimization: Non-asymptotic rates (2018). http://arxiv.org/abs/arXiv:1809.02920
14. A. K. Sahu, D. Jakovetic, D. Bajovic, S. Kar, in *IEEE EUROCON 2019 -18th International Conference on Smart Technologies*, Communication Efficient Distributed Estimation Over Directed Random Graphs (IEEE, Novi Sad, 2019), pp. 1–5. https://doi.org/10.1109/EUROCON.2019.8861544
15. D. Jakovetić, D. Bajović, A. K. Sahu, S. Kar, in *2018 IEEE Conference on Decision and Control (CDC)*, Convergence Rates for Distributed Stochastic Optimization Over Random Networks (IEEE, Miami Beach, 2018), pp. 4238–4245. https://doi.org/10.1109/CDC.2018.8619228
16. N. Krklec Jerinkić, D. Jakovetić, N. Krejić, D. Bajović, Distributed Second-Order Methods With Increasing Number of Working Nodes. IEEE Trans. Autom. Control. **65**(2), 846–853 (2020). https://doi.org/10.1109/tac.2019.2922191
17. A. Sahu, D. Jakovetić, D. Bajović, S. Kar, in *2018 IEEE Conference on Decision and Control (CDC)*, Distributed Zeroth Order Optimization Over Random Networks: A Kiefer-Wolfowitz Stochastic Approximation Approach (IEEE, Miami Beach, 2018), pp. 4951–4958. https://doi.org/10.1109/cdc.2018.8619044
18. S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Randomized gossip algorithms. IEEE/ACM Trans. Netw. **14**(SI), 2508–2530 (2006). https://doi.org/10.1109/TIT.2006.874516
19. Message Passing Interface Forum, *MPI: A Message-passing Interface Standard, Version 3.1*. (High-Performance Computing Center Stuttgart, University of Stuttgart, 2015)
20. K. I. Tsianos, S. F. Lawlor, M. G. Rabbat, in *Proceedings of the 25th International Conference on Neural Information Processing Systems, NIPS–12*, Communication/computation tradeoffs in consensus-based distributed optimization, vol. 2 (Curran Associates Inc., Red Hook, NY, USA, 2012), pp. 1943–1951
21. R. H. Byrd, S. L. Hansen, J. Nocedal, Y. Singer, A stochastic quasi-newton method for large-scale optimization. SIAM J. Optim. **26**(2), 1008–1031 (2016). https://doi.org/10.1137/140954362
22. I. A. Chen, A. Ozdaglar, in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, A fast distributed proximal-gradient method (IEEE, Monticello, 2012), pp. 601–608. https://doi.org/10.1109/Allerton.2012.6483273
23. B. Johansson, M. Rabi, M. Johansson, A randomized incremental subgradient method for distributed optimization in networked systems. SIAM J. Optim. **20**(3), 1157–1170 (2009). https://doi.org/10.1137/08073038x
24. A. Nedić, A. Olshevsky, M. G. Rabbat, Network topology and communication-computation tradeoffs in decentralized optimization. Proc. IEEE. **106**(5), 953–976 (2018). https://doi.org/10.1109/JPROC.2018.2817461
25. M. Assran, M. Rabbat, Asynchronous subgradient-push. Computing Research Repository, CoRR (2018). abs/1803.08950(2018). arXiv:1803.08950
26. M. Assran, M. Rabbat, in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, An empirical comparison of multi-agent optimization algorithms (IEEE, Montréal, 2017), pp. 573–577. https://doi.org/10.1109/GlobalSIP.2017.8309024
27. J. Zhang, K. You, AsySPA: An exact asynchronous algorithm for convex optimization over digraphs. IEEE Trans. Autom. Control. **65**(6), 2494–2509 (2020). https://doi.org/10.1109/tac.2019.2930234
28. K. I. Tsianos, S. Lawlor, M. G. Rabbat, in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning, (2012), pp. 1543–1550. https://doi.org/10.1109/Allerton.2012.6483403
29. K. Yuan, Q. Ling, W. Yin, On the convergence of decentralized gradient descent. SIAM J. Optim. **26**(3), 1835–1854 (2016). https://doi.org/10.1137/130943170
30. D. Jakovetić, J. M. F. Moura, J. Xavier, in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, Distributed nesterov-like gradient algorithms, (2012), pp. 5459–5464. https://doi.org/10.1109/CDC.2012.6425938
31. S. Sundhar Ram, A. Nedić, V. V. Veeravalli, Distributed stochastic subgradient projection algorithms for convex optimization. J. Optim. Theory Appl. **147**(3), 516–545 (2010). https://doi.org/10.1007/s10957-010-9737-7
32. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK UsersǴuide*, 3rd edn. (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 1999). https://doi.org/10.1137/1.9780898719604
33. L. Blackford, et al., An updated set of basic linear algebra subprograms (BLAS). ACM Trans. Math. Softw. **28**(2), 135–151 (2002). https://doi.org/10.1145/567806.567807
34. E. F. Tjong Kim Sang, F. De Meulder, Language-Independent Named Entity Recognition (II) (2005). https://www.clips.uantwerpen.be/conll2003/ner/. Accessed 30 May 2019
35. E. F. Tjong Kim Sang, F. De Meulder, in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003, CONLL –03*, Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition, vol. 4 (Association for Computational Linguistics, USA, 2003), pp. 142–147. https://doi.org/10.3115/1119176.1119195
36. I. Guyon, UCI Machine Learning Repository, Gisette Data Set (2008). http://archive.ics.uci.edu/ml/datasets/gisette. Accessed 29 May 2019
37. D. Dua, C. Graff, UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences (2017). http://archive.ics.uci.edu/ml. Accessed 29 May 2019
38. I. Guyon, S. Gunn, A. Ben-Hur, G. Dror, in *Proceedings of the 17th International Conference on Neural Information Processing Systems, NIPS–04*, Result analysis of the NIPS 2003 feature selection challenge, vol. 17 (MIT Press, Cambridge, MA, USA, 2004), pp. 545–552. https://eprints.soton.ac.uk/261923/
39. T. Bertin-Mahieux, UCI Machine Learning Repository, YearPredictionMSD data set (2011). https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD. Accessed 01 Sept 2019
40. T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, P. Lamere, in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, The Million Song Dataset (University of Miami, Miami, 2011), pp. 591–596. https://doi.org/10.7916/D8NZ8J07
41. Y. LeCun, C. Cortes, "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges", THE MNIST DATABASE of handwritten digits (2005). http://yann.lecun.com/exdb/mnist/. Accessed 01 Sept 2019
42. L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Proc. Mag. **29**, 141–142 (2012). https://doi.org/10.1109/MSP.2012.2211477

43. F. Graf, H.-P. Kriegel, M. Schubert, S. Poelsterl, A. Cavallaro, UCI Machine Learning Repository: Relative location of CT slices on axial axis Data Set (2011). https://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis. Accessed 08 Sept 2019
44. F. Graf, H.-P. Kriegel, M. Schubert, S. Pölsterl, A. Cavallaro, in *International Conference on Medical Image Computing and Computer-Assisted Intervention, vol. 6892*, 2d image registration in CT images using radial image descriptors (Springer, Toronto, 2011), pp. 607–614
45. UCI Machine Learning Repository, p53 Mutants Data Set. (2010; accessed on: September 03, 2019). https://archive.ics.uci.edu/ml/datasets/p53+Mutants
46. S. Danziger, R. Baronio, L. Ho, L. Hall, K. Salmon, G. Hatfield, P. Kaiser, R. Lathrop, Predicting positive p53 cancer rescue regions using Most Informative Positive (MIP) active learning. PLoS Comput. Biol. **5**, 1000498 (2009). https://doi.org/10.1371/journal.pcbi.1000498
47. S. A. Danziger, J. Zeng, Y. Wang, R. K. Brachmann, R. H. Lathrop, Choosing where to look next in a mutation sequence space: Active Learning of informative p53 cancer rescue mutants. Bioinformatics. **23**(13), 104–114 (2007). https://doi.org/10.1093/bioinformatics/btm166
48. S. Danziger, S. J. Swamidass, J. Zeng, L. Dearth, Q. Lu, J. Chen, J. Cheng, V. Hoang, H. Saigo, R. Luo, P. Baldi, R. Brachmann, R. Lathrop, Functional census of mutation sequence spaces: The example of p53 cancer rescue mutants. IEEE/ACM Trans. Comput. Biol. Bioinform. **3**, 114–25 (2006). https://doi.org/10.1109/TCBB.2006.22
49. E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002). https://doi.org/10.1007/s101070100263
50. W.-S. Zhang, "GitHub-HaidYi/admm-l1-2-logistic-regression: ADMM l1/2" logistic reression using MPI and GSL", HaidYi/admm-l1-2-logistic-regression. GitHub repository. https://github.com/HaidYi/admm-l1-2-logistic-regression. Accessed 15 May 2020

Publisher's Note