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

Lidija Fodor (lidija.fodor@dmi.uns.ac.rs)
University of Novi Sad, Faculty of sciences https://orcid.org/0000-0002-8199-7767
Dušan Jakovetić
University of Novi Sad, Faculty of sciences
Krejić Nataša
University of Novi Sad, Faculty of sciences
Nataša Krklec Jerinkić
University of Novi Sad, Faculty of sciences
Srđan Škrbić
University of Novi Sad, Faculty of sciences

Research

Keywords: Distributed optimization, High performance computing, Performance evaluation

Posted Date: August 24th, 2020

DOI: https://doi.org/10.21203/rs.3.rs-63043/v1

RESEARCH

Performance evaluation and analysis of distributed multi-agent optimization algorithms with sparsified 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
Full list of author information is available at the end of the article

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, 2, 3] and second order [4, 5, 6, 7] 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 [8], distributed control [9], vehicular networks [10], smart grid [11], etc. Relevant applications have been also demonstrated [12]. 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. The class that is considered here subsumes several existing methods [13, 14, 15, 16, 17, 18]. We also define a novel method in the same class with unidirectional communication and analyze its convergence.

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 [13]; 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 [19] 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, [20]) running on an HPC computer cluster, which is a standard and widely adopted computational system.

The rest of the paper is organized as follows. An overview of the work related to this topic is presented in Section 1.1, while Section 2 covers a few topics. We briefly describe the optimization model and the algorithmic framework for the Distributed Quasi Newton (DQN) method in Section 2.1. The algorithmic framework is described in Section 2.2. Convergence analysis of the novel method that uses unidirectional communication is presented in Section 2.3. Implementation and infrastructure are described in Section 2.4. The simulation setup is described in Section 2.5 and the proposed set of methods for the introduced algorithm is presented in Section 2.6. The results are highlighted in Section 3. 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., [12], for the dual averaging method, and [13] for the alternating direction method of multipliers. Distributed convex optimization by alternating direction method of multipliers is studied in [12]. A stochastic, efficient quasi-Newton method, using the BFGS update formula, is introduced in [21] in order to take advantage of the curvature information during approximation. 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 [28]. Finally, the focus of [29]

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 [13, 14, 15, 16, 17, 18]). 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 [13]. Also, a theoretical convergence analysis of the new method with unidirectional communication is carried out in this paper. While [18] also considers directed 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 $\lambda_2(W) < 1$, where $\lambda_1(W)$ is the largest eigenvalue of W, and $\lambda_2(W)$ 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 $x = (x_1^T, ..., x_n^T)^T \in R^{ns}$ 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

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

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 $(I - W)^{1/2}x = 0$. Further, a penalty reformulation of (3) can be stated as

$$\min_{x \in R^{ns}} F(x) + \frac{1}{2\alpha} x^T (I - W)x, \tag{4}$$

where $\frac{1}{\alpha}$ is the penalty parameter. Therefore (4) represents a quadratic penalty reformulation of the original problem (1). After standard manipulations with the penalty part we obtain

$$\min_{x \in R^{ns}} F(x) + \frac{1}{2\alpha} \sum_{i<j} W_{ij} (x_i - x_j)^2, \tag{5}$$

which is the same as (2) 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), $x^\bullet = ((x_1^\circ)^T, ..., (x_n^\circ)^T)^T$ is the solution to (2). In more details, Theorem 4 in [30] says that under Assumption 4.1, the following holds, for all $i = 1, ..., n$:

$$\begin{aligned} \|x_i^\circ - x^\star\| &\leq (\frac{\alpha LD}{1-\lambda_2(W)})\sqrt{4/c^2 - 2\alpha/c} + \frac{\alpha D}{1-\lambda_2(W)} \\ &= O(\frac{\alpha}{1-\lambda_2(W)}), \end{aligned} \tag{6}$$

$$D = \sqrt{2L(\sum_{i=1}^{n} f_i(0) - \sum_{i=1}^{n} f_i(x_i'))}; c = \frac{\mu L}{\mu + L}; \tag{7}$$

and x_i' is the minimizer of f_i.

2.2 Algorithmic framework

The algorithmic framework is presented in this Section. The framework subsumes several existing algorithms [13, 14, 15, 16, 17, 18], and it also includes a new algorithm 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.

Assume that the random variables z_i^k are independent both across nodes and across iterations. Denote by $p_k = Prob(z_i^k = 1)$, 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. With the considered algorithmic framework, solution estimate update at node i is as follows:

$$d_i^k = -\big[(M_i^k)^{-1}[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k]\big], \tag{8}$$

$$x_i^{k+1} = x_i^k + d_i^k. \tag{9}$$

Here, α is a positive parameter, known as the step-size. The values of α differs 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 $M_i^k = \alpha\nabla^2 f_i(x_i^k) + (1 - W_{ii})I$. This corresponds to the second order methods of DQN-type [13] (See ahead Section 2.5).

The pseudocode for the general algorithmic framework is in Algorithm 1. It represents the general form of the algorithm.

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 (8) – (9)
  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 3* in Section 2.5). More precisely, in this section we assume the following choice of M_i^k and ξ_{ij}^k:

$$M_i^k = I, \quad \xi_{ij}^k = z_j^k. \tag{10}$$

To the best of our knowledge, except for a different estimation setting [18], 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 $\alpha < \min\{\frac{1}{2L}, \frac{1+\lambda_n(W)}{L}\}$.

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

$$L_\Psi := L + \frac{1 - \lambda_n(W)}{\alpha}. \tag{11}$$

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

$$\alpha < \frac{2}{L_\Psi}. \tag{12}$$

Let $x^k = ((x_1^k)^T, ..., (x_n^k)^T)^T$. 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 and assume 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 $\{x^k\}$ converges to $x^\bullet$ in the expected error norm, i.e., there holds:

$$\lim_{k\to\infty} E[\|x^k - x^\bullet\|] = 0. \tag{13}$$

(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:

$$E[\|x^k - x^\bullet\|] = O(\gamma^k), \tag{14}$$

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 $\{x^k\}$ is uniformly bounded, i.e., there exists a constant $C_1 > 0$ such that $E[\|x^k\|] \leq C_1$, for all k. Then, there holds:

$$E[\|x^k - x^\bullet\|] \leq \theta^k \|x^0 - x^\bullet\| + (1 - p_{min})^2 C_2, \tag{15}$$

where $C_2 = \frac{2nC_1}{\alpha\mu}$.

Theorem 2.1 demonstrates that the Algorithm 1 with sparsified communications converges with unidirectional communications. More precisely, as long as the sequence p_k converges to one, even arbitrarily slowly', the sequence $\{x^k\}$ 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 $\{x^k\}$ 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 [13] 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 (8) – (10) is equivalent to the following:

$$x^{k+1} = x^k - \alpha[\nabla\Psi(x^k) + e^k], \tag{16}$$

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

$$e_i^k = \frac{1}{\alpha} \sum_{j \in \Omega_i} W_{ij}(z_j^k - 1)(x_i^k - x_j^k) \tag{17}$$

and $e^k \in R^n$. Indeed, in view of (10), method (8)-(9) can be represented as

$$x^{k+1} = x^k - \alpha\nabla F(x^k) - (I - W_k)x^k, \tag{18}$$

where

$$F : \mathbb{R}^n \to \mathbb{R}, F(x) = \sum_{i=1}^{n} f_i(x_i), \tag{19}$$

$$[W_k]_{ij} = \begin{cases} W_{ij} z_j^k, & \text{if } \{i,j\} \in E, i \neq j, \\ 0, & \text{if } \{i,j\} \notin E, i \neq j, \\ 1 - \sum_{l \neq i} [W_k]_{il}, & \text{if } i = j. \end{cases} \tag{20}$$

Thus,

$$\begin{aligned} x^{k+1} &= x^k - \alpha(\nabla F(x^k) + \frac{1}{\alpha}(I - W_k)x^k \pm \frac{1}{\alpha}(I - W)x^k) \\ &= x^k - \alpha(\nabla \Psi(x^k) + \frac{1}{\alpha}((I - W_k)x^k - (I - W)x^k)). \end{aligned} \tag{21}$$

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

$$e_i^k = \frac{1}{\alpha}(\sum_{j \in \Omega_i} W_{ij} z_j^k (x_i^k - x_j^k) - \sum_{j \in \Omega_i} W_{ij}(x_i^k - x_j^k)), \tag{22}$$

and (17) 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*

$$||x^k - x^\bullet|| \leq \theta^k ||x^0 - x^\bullet|| + \alpha \sum_{t=1}^{k} \theta^{k-t} ||e^{t-1}||, \tag{23}$$

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

Proof. Using (16) and the fact that $\nabla \Psi(x^\bullet) = 0$ we obtain

$$x^{k+1} - x^\bullet = x^k - x^\bullet - \alpha e^k - \alpha(\nabla \Psi(x^k) - \nabla \Psi(x^\bullet)). \tag{24}$$

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

$$\nabla \Psi(x^k) - \nabla \Psi(x^\bullet) = B_k(x^k - x^\bullet) \tag{25}$$

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

$$\|I - \alpha B_k\| \leq \max\{1 - \alpha\mu, \alpha L_\Psi - 1\} := \theta. \tag{26}$$

Notice that the Assumption 2.1 (c) implies that $\theta < 1$ since (12) holds and $L \geq \mu$. Moreover, putting together (24) - (26), we obtain

$$\|x^{k+1} - x^{\bullet}\| \leq \theta \|x^k - x^{\bullet}\| + \alpha \|e^k\| \tag{27}$$

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 $E[||x^k||] \leq C_x$ for all k, where C_x is a positive constant.*

Proof. The update rule (18) can be written equivalently as follows

$$x^{k+1} = W_k x^k - \alpha \nabla F(x^k). \tag{28}$$

Introduce $\widetilde{W_k} = W_k - W$, and rewrite (28) as

$$x^{k+1} = W x^k - \alpha \nabla F(x^k) + \widetilde{W_k} x^k. \tag{29}$$

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

$$\begin{aligned} \nabla F(x^k) - \nabla F(x') &= \underbrace{\Big[\int_0^1 \nabla^2 F(x' + t(x^k - x'))dt\Big]}_{H_k}(x^k - x') \\ &= H_k(x^k - x') = H_k x^k - H_k x', \end{aligned} \tag{30}$$

and

$$x^{k+1} = (W - \alpha H_k)x^k + \widetilde{W_k} x^k + \alpha H_k x' - \alpha \nabla F(x'). \tag{31}$$

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

$$\begin{aligned} ||x^{k+1}|| &\leq (1-\alpha\mu)||x^k|| + \underbrace{\alpha(L||x'|| + ||\nabla F(x')||)}_{c'} \\ &\quad + ||\widetilde{W}_k|| \cdot ||x_k|| \\ &= (1-\alpha\mu)||x^k|| + c' + ||\widetilde{W}_k|| \cdot ||x_k||. \end{aligned} \tag{32}$$

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

$$||\widetilde{W}_k|| \leq \sqrt{n}||\widetilde{W}_k||_1 \leq \sqrt{n}\sum_{i=1}^{n}\sum_{j=1}^{n}|[\widetilde{W}_k]_{ij}|. \tag{33}$$

Therefore,

$$||\widetilde{W}_k|| \leq 2\sqrt{n}\sum_{i=1}^{n}\sum_{j=1}^{n}W_{ij}(1-z_j^k). \tag{34}$$

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

$$E[||\widetilde{W}_k||] \leq \widetilde{C}(1-p_k) \tag{35}$$

for some positive constant $\widetilde{C}$. Now, using independence of $\widetilde{W}_k$ and x_k, the following can be obtained from (32),

$$\begin{aligned} E[||x^{k+1}||] &\leq (1-\alpha\mu)E[||x^k||] + C' + (1-p_{k+1})\widetilde{C}E[||x^k||] \\ &= (1-\alpha\mu+\widetilde{C}(1-p_{k+1}))E[||x^k||] + C'. \end{aligned} \tag{36}$$

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

$$E[||x^{k+1}||] \leq (1-\frac{1}{2}\alpha\mu)E[||x^k||] + C'. \tag{37}$$

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

Applying Lemma 2.3, the following result is obtained.

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

$$E[||e^k||] \leq (1 - p_k)C_e, \tag{38}$$

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

Proof. The proof follows straightforwardly from (17) and Lemma 2.3. Consider (22). Then, $|e_i^k|$ can be upper bounded as follows:

$$|e_i^k| \leq \frac{1}{\alpha} \sum_{j \in \Omega_i} w_{ij}|1 - z_j^k|2\|x^k\|. \tag{39}$$

This yields:

$$\|e^k\| \leq \|e^k\|_1 = \sum_{i=1}^{n} \frac{2}{\alpha} \sum_{j \in \Omega_i} w_{ij}|1 - z_j^k|\|x^k\|. \tag{40}$$

Taking expectation while using independence of z_j^k and x^k, and using $E(\|x^k\|) \leq C_1$; $\sum_{j \in \Omega_i} \leq 1$; and $E(|1 - z_j^k|) = 1 - p_k$, 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, the following can be obtained

$$\begin{aligned} E[||x^k - x^\bullet||] &\leq \theta^k||x^0 - x^\bullet|| + \alpha \sum_{t=1}^{k} \theta^{k-t} E[||e^{t-1}||] \\ &\leq \theta^k||x^0 - x^\bullet|| + \alpha \sum_{t=1}^{k} \theta^{k-t} \cdot C_e(1 - p_{t-1}). \end{aligned} \tag{41}$$

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

$$E[\|x^k - x^\bullet\|] \to 0, \tag{42}$$

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 (41) to this choice of p_k, the following holds

$$E[||x^k - x^\bullet||] \leq \theta^k ||x^0 - x^\bullet|| + \alpha C_e \sum_{t=1}^{k} \theta^{k-t} \delta^t, \tag{43}$$

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 [13]), we obtain the result.

Finally, we prove part (c). Here, we upper bound the term $(1 - p_{t-1})$ in (41) with $(1 - p_{min})$. For this case we obtain

$$\begin{aligned} E[||x^k - x^\bullet||] \leq \theta^k ||x^0 - x^\bullet|| \\ + (1 - p_{min}) C_e \frac{1}{\mu}, \end{aligned} \tag{44}$$

which completes the proof of part (c).□

2.4 Implementation and infrastructure

A parallel implementation of Algorithm 1 was developed, using MPI [20]. 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 [12]. 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 $||\nabla\Psi(x^k)|| \leq \epsilon$, where $\epsilon = 0.01$. Note that the gradient $\nabla\Psi(x^k)$ is not computable by any node in a distributed graph G in general. In our implementation $\nabla\Psi(x^k)$ 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

$$f_i(x) = \sum_{j=1}^{J} \mathcal{J}_{logis}(b_{ij}(x_1^\top a_{ij} + x_0)) + \frac{\tau}{n}||x||^2. \tag{45}$$

Here, $x = (x_1^T, x_0) \in \mathbb{R}^{s-1} \times \mathbb{R}$ 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 (47) - (64)).

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, 37, 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 [39, 40, 37]. 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 [43, 37, 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 [45, 37, 46, 47, 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 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 3 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 $M_i^k = \alpha \nabla^2 f_i(x_i^k) + (1 - W_{ii})I$, 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 (8) pays off. With $M_i^k = I$ we are performing (probably larger) number of cheaper iterations.

2.6 Description of the methods

When considering the solution update:

$$x_i^{k+1} = x_i^k + d_i^k, \tag{46}$$

the following alternatives of Algorithm 1 are considered.

- ***Method 0***: The initial version of the algorithm, used as the benchmark here, without sparsification (all the nodes are active all the time), the communication is always bidirectional, and the Hessian is included in the computation, so it is a second order method. More precisely, the method is defined by the following. For all $i = 1, \ldots, n$, given x_i^k, we have

$$\xi_{i,j}^k = 1, p_k = 1, M_i^k = \alpha \nabla^2 f_i(x_i^k) + (1 - W_{ii})I, \tag{47}$$

$$d_i^k = -\big[(M_i^k)^{-1}[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)]\big]. \tag{48}$$

- ***Method 1***: Bidirectional communication, with increasing communication probability. Here, the Hessian approximation is replaced with the identity matrix, resulting in the following first order method:

$$\xi_{i,j}^k = z_i^k \cdot z_j^k, p_k = 1 - 0.5^k, M_i^k = I, \tag{49}$$

$$d_i^k = -\big[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k\big]. \tag{50}$$

- ***Method 2***: Bidirectional communication, with decreasing communication probability and first order updates,

$$\xi_{i,j}^k = z_i^k \cdot z_j^k, p_k = (k+1)^{-1}, M_i^k = I, \tag{51}$$

$$d_i^k = -\big[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k\big]. \tag{52}$$

- ***Method 3***: Unidirectional communication, with increasing communication probability and first order method updates,

$$\xi_{i,j}^k = z_j^k, p_k = 1 - 0.5^k, M_i^k = I, \tag{53}$$

$$d_i^k = -\big[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k\big]. \tag{54}$$

- ***Method 4***: Unidirectional communication with decreasing communication probability and first order updates,

$$\xi_{i,j}^k = z_j^k, p_k = (k+1)^{-1}, M_i^k = I, \tag{55}$$

$$d_i^k = -\big[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k\big]. \tag{56}$$

- ***Method 5***: Bidirectional communication, with increasing communication probability and second order updates,

$$\xi_{i,j}^k = z_i^k \cdot z_j^k, p_k = 1 - 0.5^k, M_i^k = \alpha \nabla^2 f_i(x_i^k) + (1 - W_{ii})I, \tag{57}$$

$$d_i^k = -\big[(M_i^k)^{-1}[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k]\big]. \tag{58}$$

- ***Method 6***: Bidirectional communication, with decreasing communication probability and second order updates,

$$\xi_{i,j}^k = z_i^k \cdot z_j^k, p_k = (k+1)^{-1}, M_i^k = \alpha \nabla^2 f_i(x_i^k) + (1 - W_{ii})I, \tag{59}$$

$$d_i^k = -\big[(M_i^k)^{-1}[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k]\big]. \tag{60}$$

- ***Method 7***: Unidirectional communication, with increasing communication probability and second order updates,

$$\xi_{i,j}^k = z_j^k, p_k = 1 - 0.5^k, M_i^k = \alpha \nabla^2 f_i(x_i^k) + (1 - W_{ii})I, \tag{61}$$

$$d_i^k = -\big[(M_i^k)^{-1}[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k]\big]. \tag{62}$$

- ***Method 8***: Unidirectional communication, decreasing communication probability and second order updates,

$$\xi_{i,j}^k = z_j^k, p_k = (k+1)^{-1}, M_i^k = \alpha \nabla^2 f_i(x_i^k) + (1 - W_{ii})I, \tag{63}$$

$$d_i^k = -\big[(M_i^k)^{-1}[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k]\big]. \tag{64}$$

- ***Method 9***: Bidirectional communication without communication sparsification. This is a first order method. It corresponds to *Method 0* without second order information.

$$\xi_{i,j}^k = 1, p_k = 1, M_i^k = I, \tag{65}$$

$$d_i^k = -\big[\alpha \nabla f_i(x_i^k) + \sum_{j \epsilon \Omega_i} W_{ij}(x_i^k - x_j^k)\xi_{i,j}^k\big]. \tag{66}$$

Note that *Methods 1-8* 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., [13], [18]. 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 0* and *Method 9*), 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 3* is presented here, where *Methods 4, 7* and *8*, that also rely on unidirectional communication, remain open for theoretical analysis. The *Methods 1, 2, 5* and *6*, using bidirectional communication are already analysed in the literature (see [13, 14, 15, 16, 17, 18]).

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. These tests are performed using the data sets Conll and CT with *Method 0*. After that, we perform a sequence of tests using *Methods 0-9* 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.

Fig. 1 and 2 represent a performance comparison between the executions of the algorithm using different d-regular and grid graphs with *Method 0* 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 when increasing the 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.

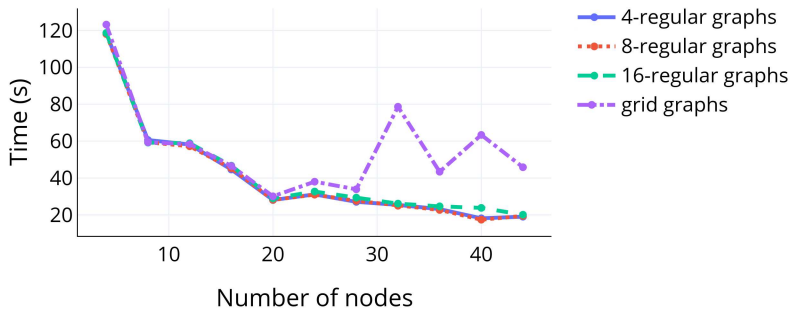

Figure 1 Comparing the execution time using regular and grid graphs, CT data set, for *Method 0*

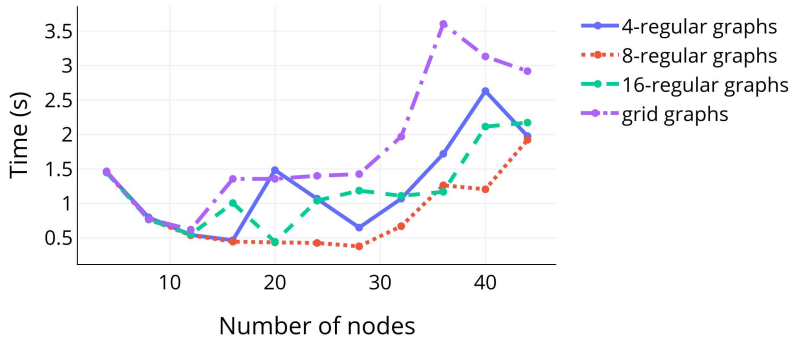

Figure 2 Comparing the execution time using regular and grid graphs, Conll data set, for *Method 0*

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. Fig. 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 $n < 8$.

Table 1 lists the execution time for each of the 10 methods, i.e. *Methods 0-9*, 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

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

Method	Execution time (s)
Method 0	9661.42
Method 1	4.64
Method 2	1.89
Method 3	6.04
Method 4	3.56
Method 5	43126.71
Method 6	22683.84
Method 7	22029.20
Method 8	9651.77
Method 9	3.16

Table 2 Execution time for different variations of Algorithm 1, for 12 nodes, Mnist data set

Method	Execution time (s)
Method 0	19045.00
Method 1	336.31
Method 2	118.16
Method 3	353.31
Method 4	342.59
Method 5	3124.56
Method 6	11853.99
Method 7	12259.79
Method 8	N/A
Method 9	161.33

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 2* has the best performance. When comparing *Method 9* to *Method 0*, 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 2* 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 4*, using unidirectional communication and decreasing communication probability performs better than *Method 1*, with bidirectional and 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.

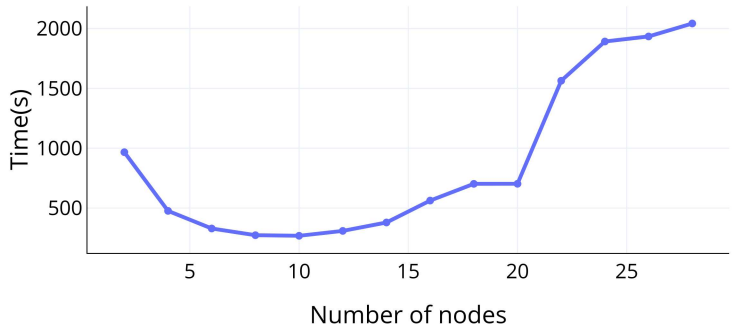

Figure 3 Scaling properties of *Method 1*, for the YearPredictionMSD data set

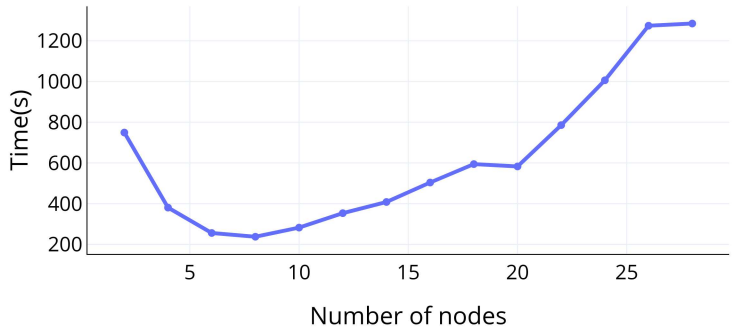

Figure 4 Scaling properties of *Method 3*, for the Mnist data set

As the nature of the data can highly influence the results, let us consider another example. Table 2 also contains the execution time for each of the 10 algorithms with 12 nodes for the Mnist data set (*Method 8* does not converge for the given execution time limit). The behaviour of this data set differs from the p53 data set, observed in Table 1. For example, for 12 nodes *Method 2* 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 2 are very similar to those from Table 1. In fact, it seems that the properties of particular methods are similar as long as the data sets are of similar volume.

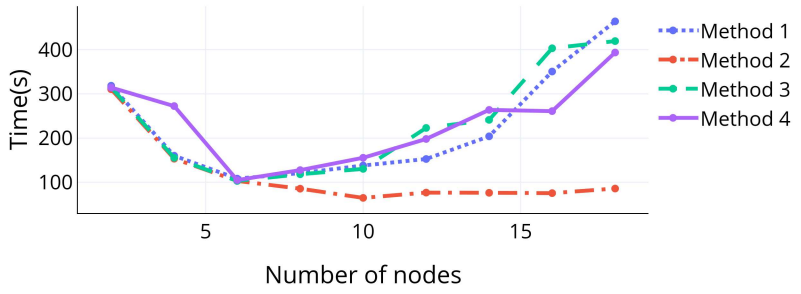

Figure 5 Execution times for the first order *Methods 1-4* on CT data set

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. Fig. 3 and Fig. 4 represent examples of the scaling properties of the algorithm, for *Method 1* on the YearPredictionMSD data set and for *Method 3* 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.

Fig. 5 and Fig. 6 represent the execution times for first order methods with communication sparsification, i.e. *Methods 1-4* for the CT and Gisette data set, respectively. From Fig. 5, it can be concluded that the optimal number of nodes for *Methods 1, 3* and *4*, is the same value $n = 6$. However, *Method 2* 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 1, 3* and *4* is again the same, $n = 8$. *Method 2* also performs differently here, with lower execution time values, compared to other

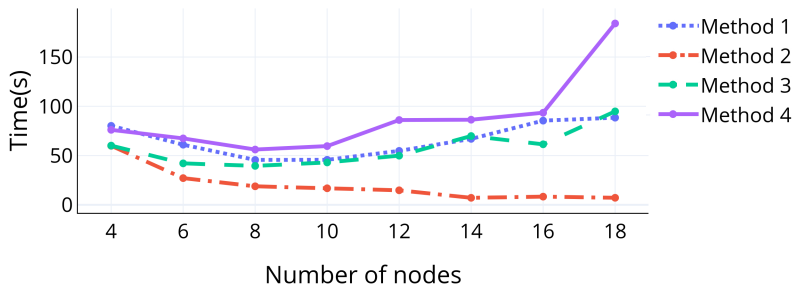

Figure 6 Execution times for the first order *Methods 1-4* on Gisette data set

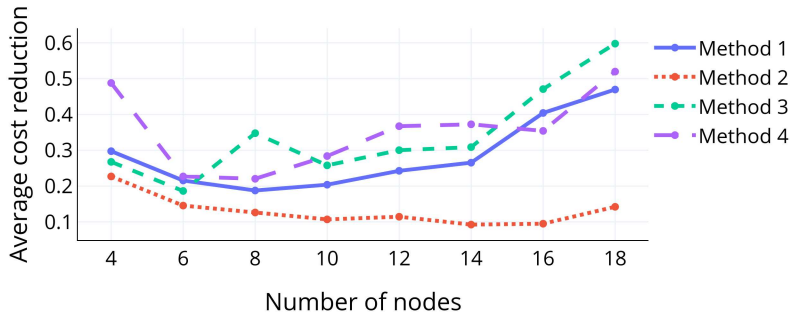

Figure 7 Average cost reduction compared to the worst relevant tested method, *Methods 1, 2, 3, 4*

first order methods. The optimal number of nodes for the second order methods tends to be a larger 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.

Fig. 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 1, 2, 3* and *4*. 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 Fig. 5 and Fig. 6. *Method 2* has the best

Table 3 Percentages of successful test with respect to the overall number of tests

Method	Percentage
Method 0	98.3
Method 1	99.1
Method 2	100
Method 3	100
Method 4	100
Method 5	84.1
Method 6	95.8
Method 7	95.8
Method 8	35
Method 9	100

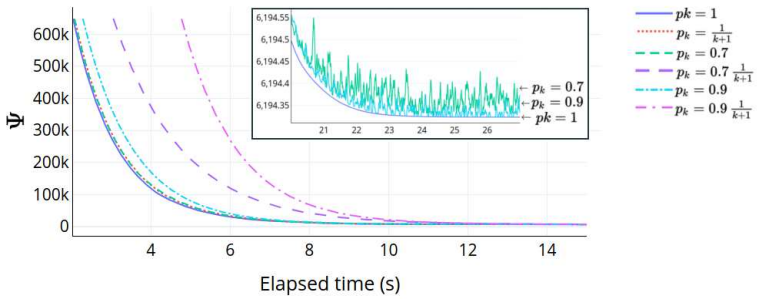

Figure 8 Comparison between using different values of $p_k \leq 1$, first order method, unidirectional communication, Conll data set

performance properties. Also, for each method, an optimal number of nodes can be identified.

Table 3 shows the percentages of successful tests for all methods, i.e., of tests that satisfy the stopping criteria $\|\Phi(x^k)\| < 0.01$ within maximal execution time of 15 hours. In the failed tests the iterations are also approaching the solution, but they did not reach it within the given time limit. The results indicate that the first order methods are better choice in this environment as *Method 8* 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 in order to ensure convergence during the time consuming iterations.

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. Fig. 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.

Fig. 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.

Fig. 9 shows the performance profile for all the test on all data sets for the 10 methods, where Fig. 10 and Fig. 11 display the performance profile for first and second order methods, respectively. Fig. 9 and Fig. 10 identify *Method 2* as the best choice within the framework for Algorithm 1. Observing the methods without sparsification, i.e. *Methods 0* and *9*, Fig. 9 indicates that the first order method, *Method 9*, performs better than the second order method, *Method 0*. The same is true if we consider the methods with sparsification. Considering methods with decreasing communication probability and using bidirectional communication, *Method 2* performs clearly much better than *Method 6*. When comparing the other first and second order methods using the same sparsification (*Method 1* and *Method 5*, *Method 3* and *Method 7*, *Method 4* and *Method 8*), first order methods performs better in 61% of test cases. Also, the convergence rate for first order methods is higher (See Table 3). 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 2*. Generally, the best performing method is a first order method using the appropriate sparsification (bidirectional with decreasing communication probability), *Method 2*.

Fig. 12 represents the performance profile for the tests on the Gisette data set. Here, *Method 2* can be also identified as the most suitable, followed by *Method 9*, and later by *Method 3*, *Method 1* and *Method 4*, 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. Fig. 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 2* 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 $r = 60000$ 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 7* 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 5* and *7*) perform better (for both unidirectional and bidirectional communication). With the decreasing communication probability the first order methods (*Methods 2* and *4*) give better results.

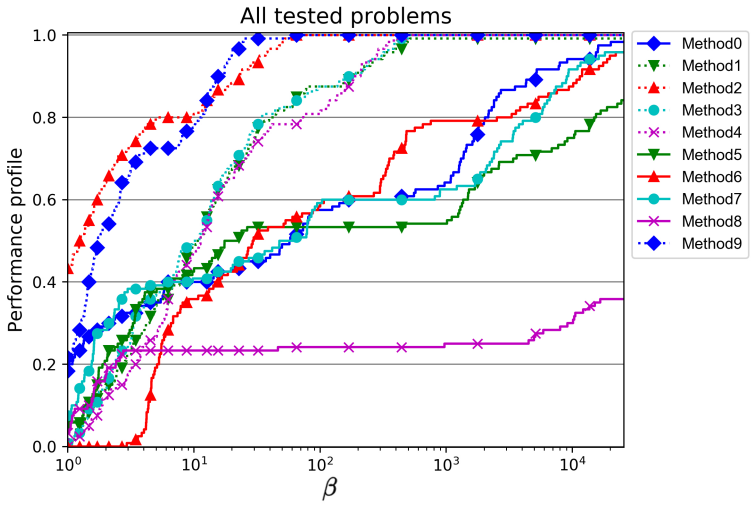

Figure 9 Performance profile for the all 10 methods, based on all the performed tests

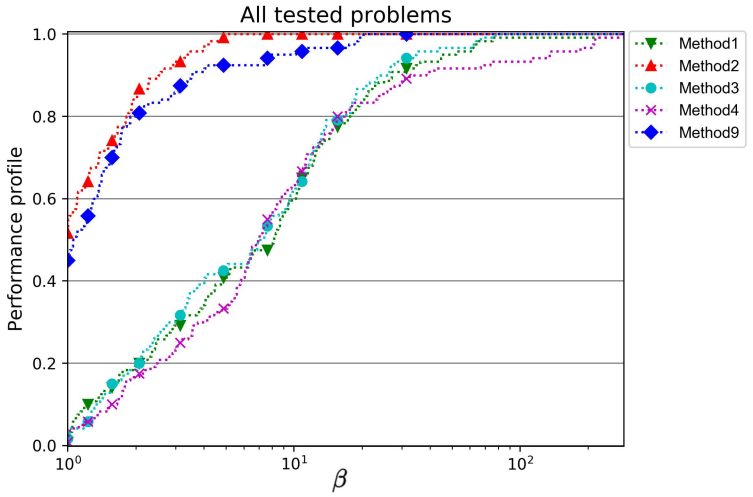

Figure 10 Performance profile for first order *Methods 1, 2, 3, 4, 9*, for all performed tests

Fig. 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.

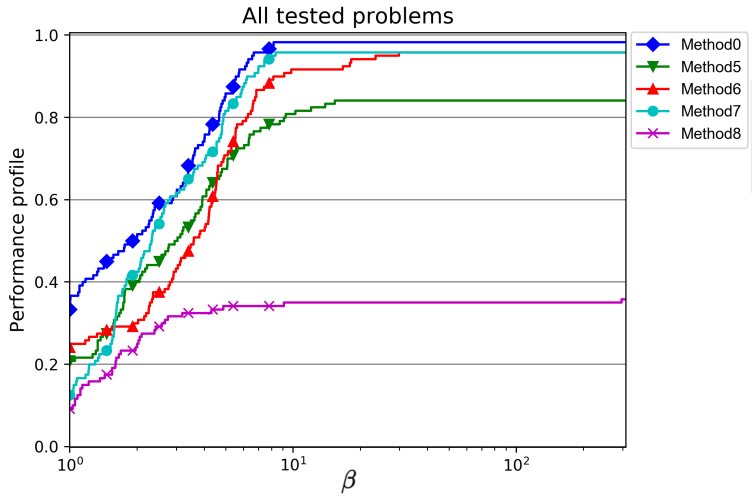

Figure 11 Performance profile for second order *Methods 0, 5, 6, 7, 8*, for all performed tests

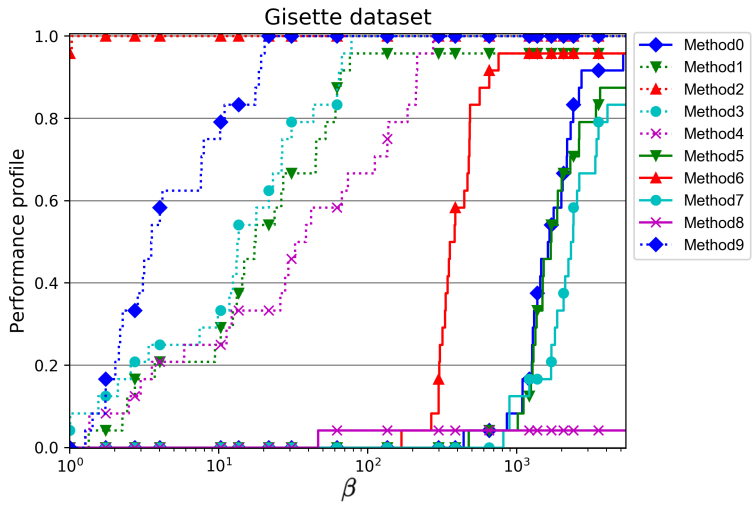

Figure 12 Performance profile for all 10 methods, for the tests performed on the Gisette data set

Table 4 Comparison of the second order *Methods 0* and *5* with ADMM

Method	Execution time
ADMM	4.487
Method 0	0.247
Method 5	0.226

As problem (1) can be solved using the Alternating Direction Method of Multipliers (ADMM) [12], we compared Algorithm 1 to an ADMM implementation for logistic regression [50], on the Conll data set. More precisely, the method in [12] solves problem (1) assuming the presence of a central node that communicates to

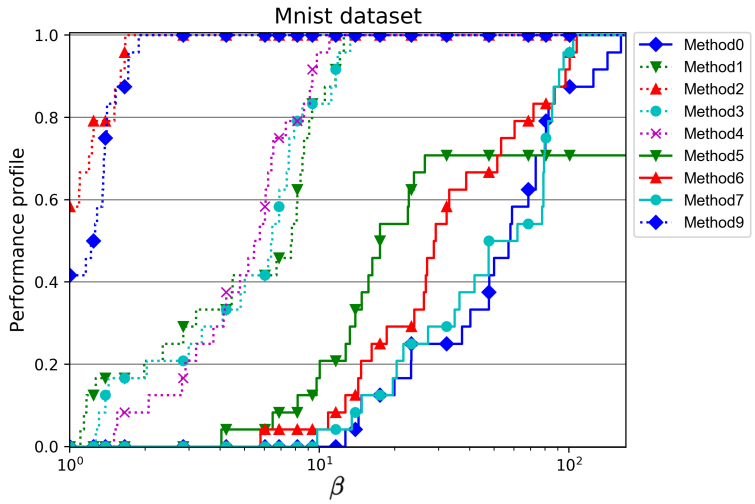

Figure 13 Performance profile for all 10 methods, for the tests performed on the Mnist data set

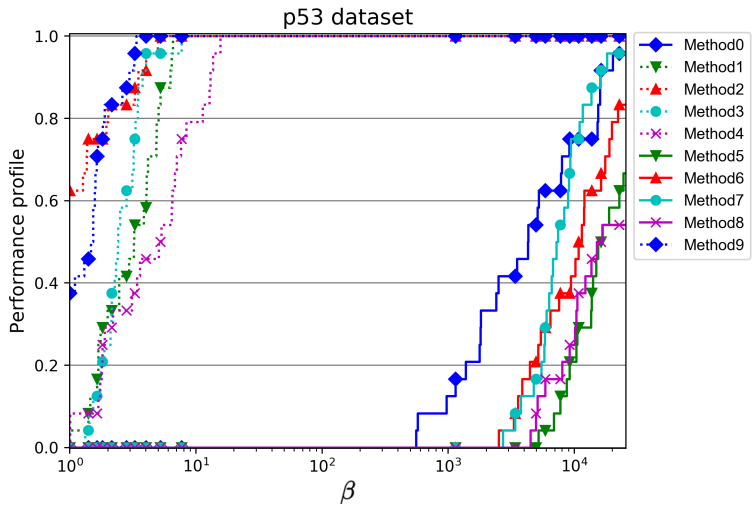

Figure 14 Performance profile for all 10 methods, for the tests performed on the p53 data set

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 0* and *5* and ADMM is shown in Table 4. We calculate the value of $\Phi^k = \frac{1}{n}\sum_{i=1}^{n} f(x_i^k)$, 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 4 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

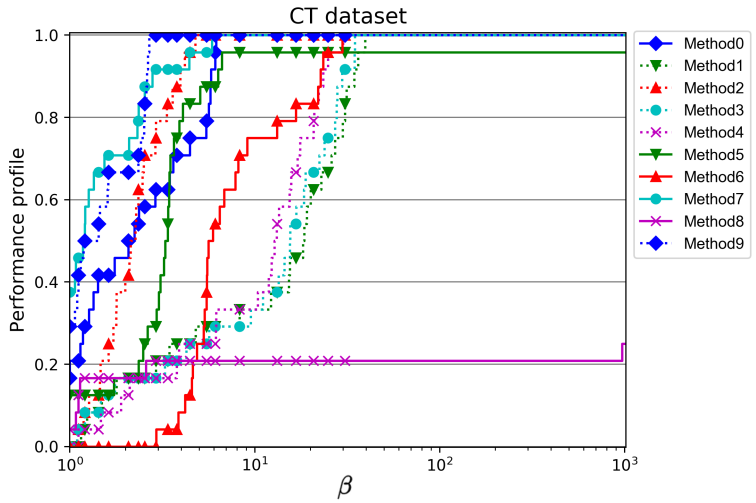

Figure 15 Performance profile for all 10 methods, for the tests performed on the CT data set

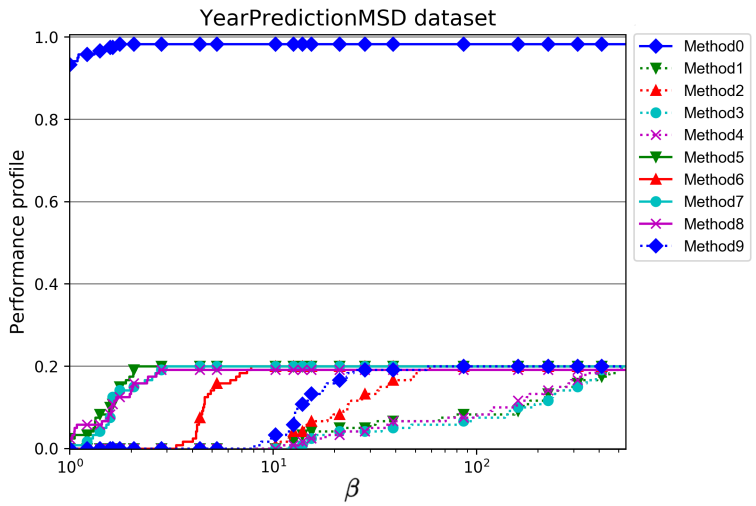

Figure 16 Performance profile for all 10 methods, for the tests perfomed on the YearPredictionMSD data set

for this comparison is the following. *Methods 0-9* 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 *Methods 0-9* and finally included the results for the best performing second order methods, i.e. *Methods 0* and *5*. More precisely, *Method 5* (a second order method with sparsification) is here the best performing method across all *Meth-*

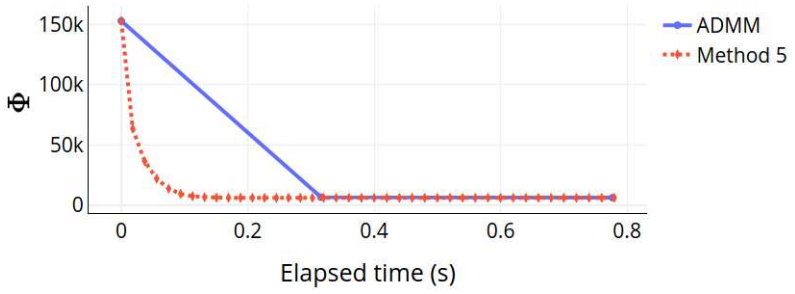

Figure 17 The comparison between ADMM and Method 5 on Conll data set

ods 0-9, while *Method 0* 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. Fig. 17 shows the comparison between Method 5 and ADMM. *Method 5* takes a larger number of significantly faster iterations, compared to ADMM, and hence results with shorter execution time needed to approach Φ^*.

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 [13, 14] 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 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 2*) 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.

Declarations

Abbrevations

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

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, 46, 47, 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 THE MNIST DATABASE of Handwritten Digits web site [http://yann.lecun.com/exdb/mnist/] [41, 42] (the Mnist data set).

Competing interests

The authors declare that they have no competing interests.

Funding

Not applicable.

Author's 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).

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.

References

1. Nedic, A., Ozdaglar, A.: Distributed subgradient methods for multi-agent optimization. IEEE Transactions on Automatic Control **54**(1), 48–61 (2009). doi:10.1109/tac.2008.2009515
2. Ram, S.S., Nedich, A., Veeravalli, V.V.: Distributed stochastic subgradient projection algorithms for convex optimization. Journal of Optimization Theory and Applications **147**(3), 516–545 (2010). doi:10.1007/s10957-010-9737-7
3. Jakovetic, D., Xavier, J.M.F., Moura, J.M.F.: Fast distributed gradient methods. IEEE Transactions on Automatic Control **59**(5), 1131–1146 (2014). doi:10.1109/tac.2014.2298712
4. Mokhtari, A., Ling, Q., Ribeiro, A.: Network newton distributed optimization methods. IEEE Transactions on Signal Processing **65**(1), 146–161 (2017). doi:10.1109/tsp.2016.2617829
5. Bajović, D., Jakovetić, D., Krejić, N., Krklec Jerinkić, N.: Newton-like method with diagonal correction for distributed optimization. SIAM Journal on Optimization **27**(2), 1171–1203 (2017). doi:110.1137/15m1038049
6. Mokhtari, A., Ling, Q., Ribeiro, A.: Network newton-part i: Algorithm and convergence (2015). arXiv:1504.06017
7. Mokhtari, A., Ling, Q., Ribeiro, A.: Network newton-part ii: Convergence rate and implementation (2015). arXiv:1504.06020
8. Zhang, K., Yang, Z., Liu, H., Zhang, T., Basar, T.: Fully decentralized multi-agent reinforcement learning with networked agents (2018). arXiv:1802.08757
9. Shamma, J.: Cooperative Control of Distributed Multi-Agent Systems. Wiley-Interscience, USA (2008). doi:10.1002/9780470724200
10. Salkham, A., Cunningham, R., Garg, A., Cahill, V.: A collaborative reinforcement learning approach to urban traffic control optimization. In: 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. 2, pp. 560–566. IEEE, Sydney, NSW, Australia (2008). doi:10.1109/WIIAT.2008.88
11. Roche, R., Blunier, B., Miraoui, A., Hilaire, V., Koukam, A.: Multi-agent systems for grid energy management: A short review. In: IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society, pp. 3341–3346 (2010). doi:10.1109/IECON.2010.5675295
12. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning **3**(1), 1–122 (2011). doi:10.1561/2200000016
13. Lrklec Jerinkić, N., Jakovetić, D., Krejić, N., Bajović, D.: Distributed second-order methods with increasing number of working nodes. IEEE Transactions on Automatic Control **65**(2), 846–853 (2020). doi:10.1109/tac.2019.2922191
14. Jakovetić, D., Bajović, D., Krejić, N., Krklec Jerinkić, N.: Distributed gradient methods with variable number of working nodes. IEEE Transactions on Signal Processing **64**(15), 4080–4095 (2016). doi:10.1109/TSP.2016.2560133
15. Jakovetić, D., Bajović, D., Sahu, A.K., Kar, S.: Convergence rates for distributed stochastic optimization over random networks. In: 2018 IEEE Conference on Decision and Control (CDC), Miami Beach, FL, USA, pp. 4238–4245 (2018). doi:10.1109/CDC.2018.8619228
16. Sahu, A., Jakovetić, D., Bajović, D., Kar, S.: Distributed zeroth order optimization over random networks: A kiefer-wolfowitz stochastic approximation approach. In: 2018 IEEE Conference on Decision and Control (CDC), Miami Beach, FL, USA, pp. 4951–4958 (2018). doi:10.1109/cdc.2018.8619044
17. Sahu, A.K., Jakovetic, D., Bajovic, D., Kar, S.: Communication-efficient distributed strongly convex stochastic optimization: Non-asymptotic rates (2018). arXiv:1809.02920
18. Sahu, A.K., Jakovetic, D., Bajovic, D., Kar, S.: Communication efficient distributed estimation over directed random graphs. In: IEEE EUROCON 2019 -18th International Conference on Smart Technologies, Novi Sad, Serbia, pp. 1–5 (2019). doi:10.1109/EUROCON.2019.8861544
19. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. IEEE/ACM Trans. Netw. **14**(SI), 2508–2530 (2006). doi:10.1109/TIT.2006.874516
20. Message Passing Interface Forum: MPI: A Message-passing Interface Standard, Version 3.1.

High-Performance Computing Center Stuttgart, University of Stuttgart (2015)

21. Byrd, R.H., Hansen, S.L., Nocedal, J., Singer, Y.: A stochastic quasi-newton method for large-scale optimization. SIAM Journal on Optimization **26**(2), 1008–1031 (2016). doi:10.1137/140954362
22. Chen, I.A., Ozdaglar, A.: A fast distributed proximal-gradient method. In: 2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Monticello, IL, USA, pp. 601–608 (2012). doi:10.1109/Allerton.2012.6483273
23. Johansson, B., Rabi, M., Johansson, M.: A randomized incremental subgradient method for distributed optimization in networked systems. SIAM Journal on Optimization **20**(3), 1157–1170 (2009). doi:10.1137/08073038x
24. Nedić, A., Olshevsky, A., Rabbat, M.G.: Network topology and communication-computation tradeoffs in decentralized optimization. Proceedings of the IEEE **106**(5), 953–976 (2018). doi:10.1109/JPROC.2018.2817461
25. Assran, M., Rabbat, M.: Asynchronous subgradient-push. CoRR **abs/1803.08950** (2018). arXiv:1803.08950
26. Assran, M., Rabbat, M.: An empirical comparison of multi-agent optimization algorithms. In: 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP), pp. 573–577 (2017). doi:10.1109/GlobalSIP.2017.8309024. IEEE
27. Zhang, J., You, K.: Asyspa: An exact asynchronous algorithm for convex optimization over digraphs. IEEE Transactions on Automatic Control **65**(6), 2494–2509 (2020). doi:10.1109/tac.2019.2930234
28. Tsianos, K.I., Lawlor, S.F., Rabbat, M.G.: Communication/computation tradeoffs in consensus-based distributed optimization. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2. NIPS'12, pp. 1943–1951. Curran Associates Inc., Red Hook, NY, USA (2012)
29. Tsianos, K.I., Lawlor, S., Rabbat, M.G.: Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In: 2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 1543–1550 (2012). doi:10.1109/Allerton.2012.6483403
30. Yuan, K., Ling, Q., Yin, W.: On the convergence of decentralized gradient descent. SIAM Journal on Optimization **26**(3), 1835–1854 (2016). doi:10.1137/130943170
31. Sundhar Ram, S., Nedić, A., Veeravalli, V.V.: Distributed stochastic subgradient projection algorithms for convex optimization. Journal of Optimization Theory and Applications **147**(3), 516–545 (2010). doi:10.1007/s10957-010-9737-7
32. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK UsersǴuide, 3rd edn. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA (1999). doi:10.1137/1.9780898719604
33. Blackford, L., et al.: An updated set of basic linear algebra subprograms (blas). ACM Trans. Math. Softw. **28**(2), 135–151 (2002). doi:10.1145/567806.567807
34. Tjong Kim Sang, E.F., De Meulder, F.: Language-Independent Named Entity Recognition II (2005; accessed on: May 30, 2019). https://www.clips.uantwerpen.be/conll2003/ner/
35. Tjong Kim Sang, E.F., De Meulder, F.: Introduction to the conll-2003 shared task: Language-independent named entity recognition. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4. CONLL '03, pp. 142–147. Association for Computational Linguistics, USA (2003). doi:10.3115/1119176.1119195
36. UCI Machine Learning Repository: Gisette Data Set (2008; accessed on: May 29, 2019). http://archive.ics.uci.edu/ml/datasets/gisette
37. Dua, D., Graff, C.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences (2017). http://archive.ics.uci.edu/ml
38. Guyon, I., Gunn, S., Ben-Hur, A., Dror, G.: Result analysis of the nips 2003 feature selection challenge. In: Proceedings of the 17th International Conference on Neural Information Processing Systems. NIPS'04, vol. 17, pp. 545–552. MIT Press, Cambridge, MA, USA (2004). https://eprints.soton.ac.uk/261923/
39. UCI Machine Learning Repository: YearPredictionMSD data set (2011; accessed on: September 01, 2019). https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD

40. Bertin-Mahieux, T., Ellis, D.P.W., Whitman, B., Lamere, P.: The million song dataset. In: Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011) (2011). doi:10.7916/D8NZ8J07
41. LeCun, Y., Cortes, C.: THE MNIST DATABASE of handwritten digits (2005; accessed on: September 01, 2019). http://yann.lecun.com/exdb/mnist/
42. Deng, L.: The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine **29**, 141–142 (2012). doi:10.1109/MSP.2012.2211477
43. UCI Machine Learning Repository: Relative location of CT slices on axial axis Data Set (2011; accessed on: September 08, 2019). https://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis
44. Graf, F., Kriegel, H.-P., Schubert, M., Pölsterl, S., Cavallaro, A.: 2d image registration in ct images using radial image descriptors. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, vol. 6892, pp. 607–614. Springer, ??? (2011). doi:10.1007/978-3-642-23629-7$_74.Springer$
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. Danziger, S., Baronio, R., Ho, L., Hall, L., Salmon, K., Hatfield, G., Kaiser, P., Lathrop, R.: Predicting positive p53 cancer rescue regions using most informative positive (mip) active learning. PLoS computational biology **5**, 1000498 (2009). doi:10.1371/journal.pcbi.1000498
47. Danziger, S.A., Zeng, J., Wang, Y., Brachmann, R.K., Lathrop, R.H.: Choosing where to look next in a mutation sequence space: Active Learning of informative p53 cancer rescue mutants. Bioinformatics **23**(13), 104–114 (2007). doi:10.1093/bioinformatics/btm166
48. Danziger, S., Swamidass, S.J., Zeng, J., Dearth, L., Lu, Q., Chen, J., Cheng, J., Hoang, V., Saigo, H., Luo, R., Baldi, P., Brachmann, R., Lathrop, R.: Functional census of mutation sequence spaces: The example of p53 cancer rescue mutants. IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM **3**, 114–25 (2006). doi:10.1109/TCBB.2006.22
49. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical Programming **91**(2), 201–213 (2002). doi:10.1007/s101070100263
50. ADMM l1 and l2 logistic regression. GitHub (accessed on: May 15, 2020). https://github.com/HaidYi/admm-l1-2-logistic-regression

Figures

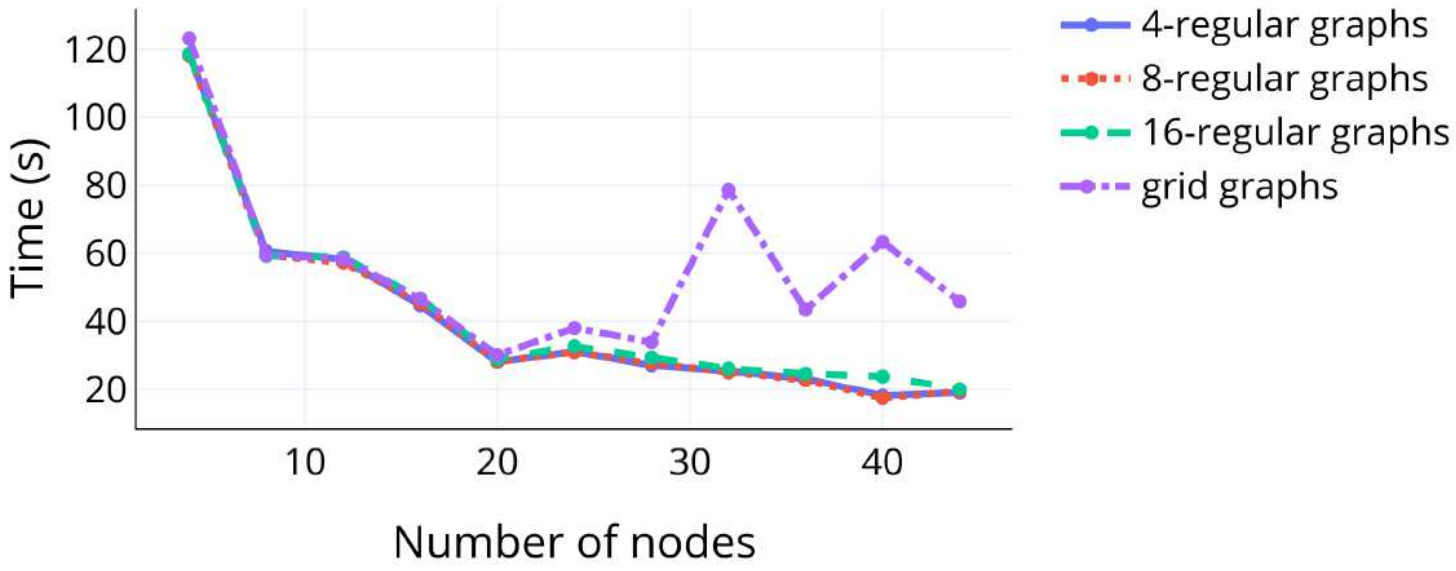

Figure 1

Comparing the execution time using regular and grid graphs, CT data set, for Method 0

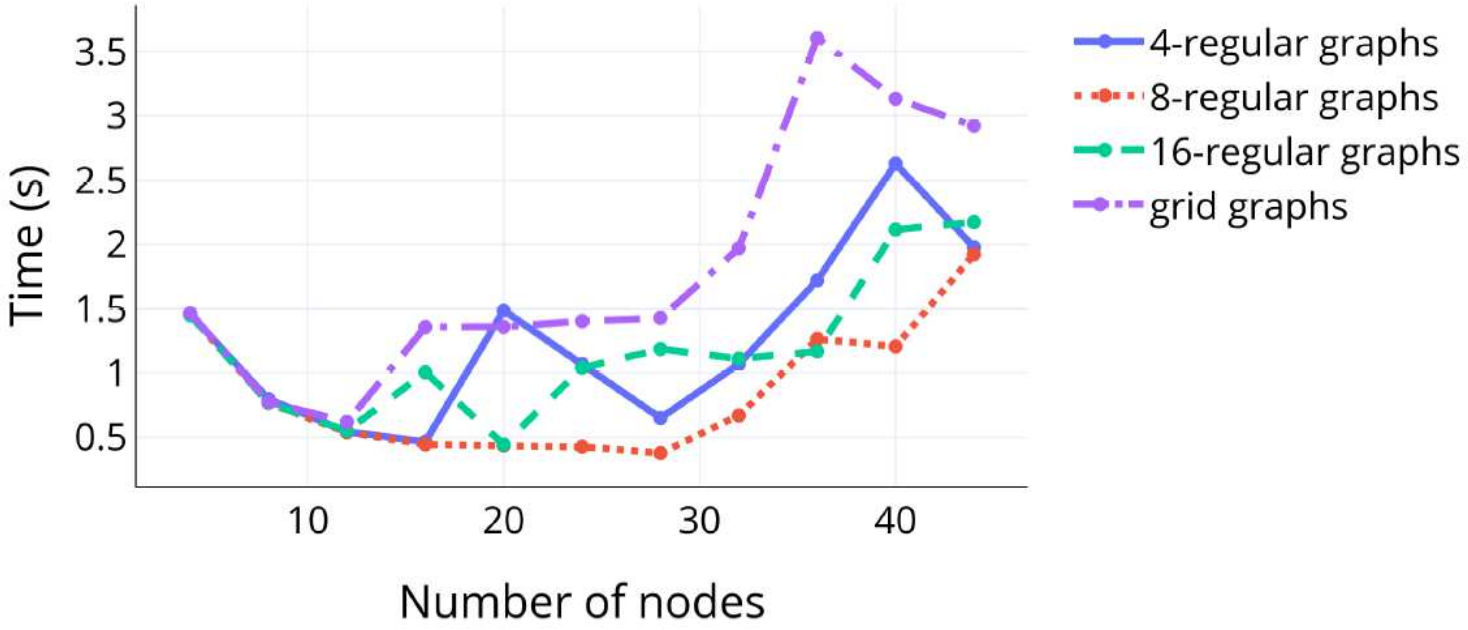

Figure 2

Comparing the execution time using regular and grid graphs, Conll data set, for Method 0

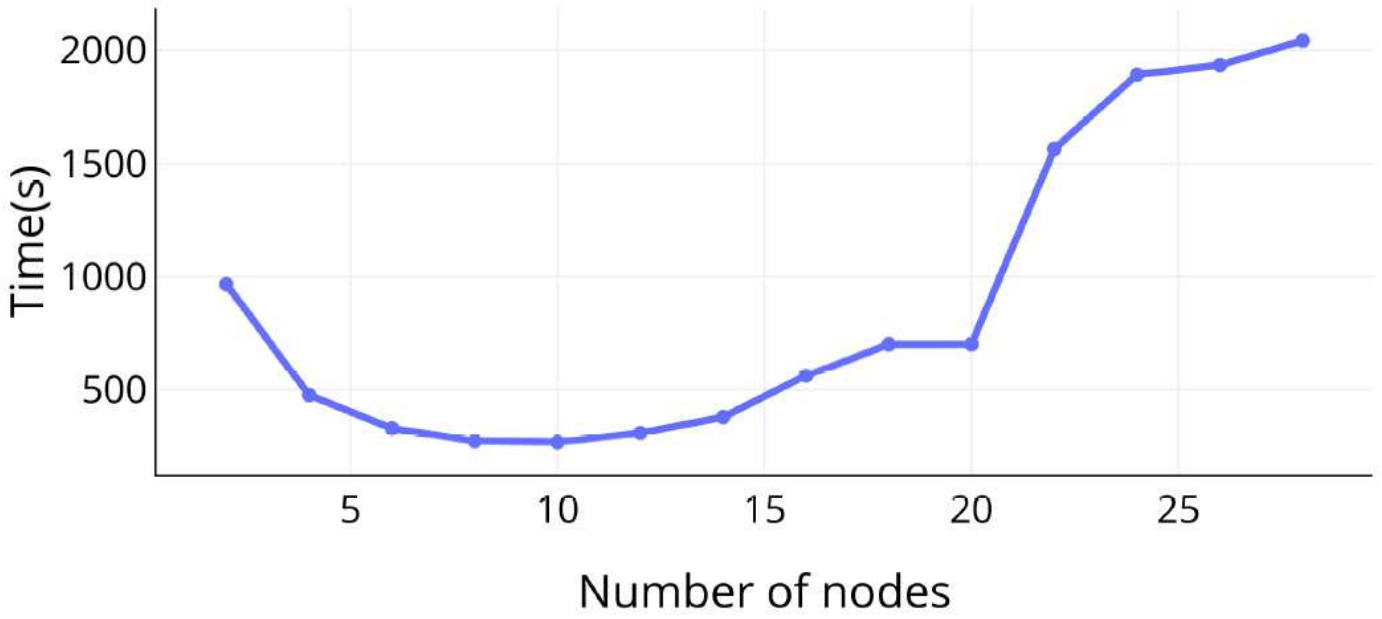

Figure 3

Scaling properties of Method 1, for the YearPredictionMSD data set

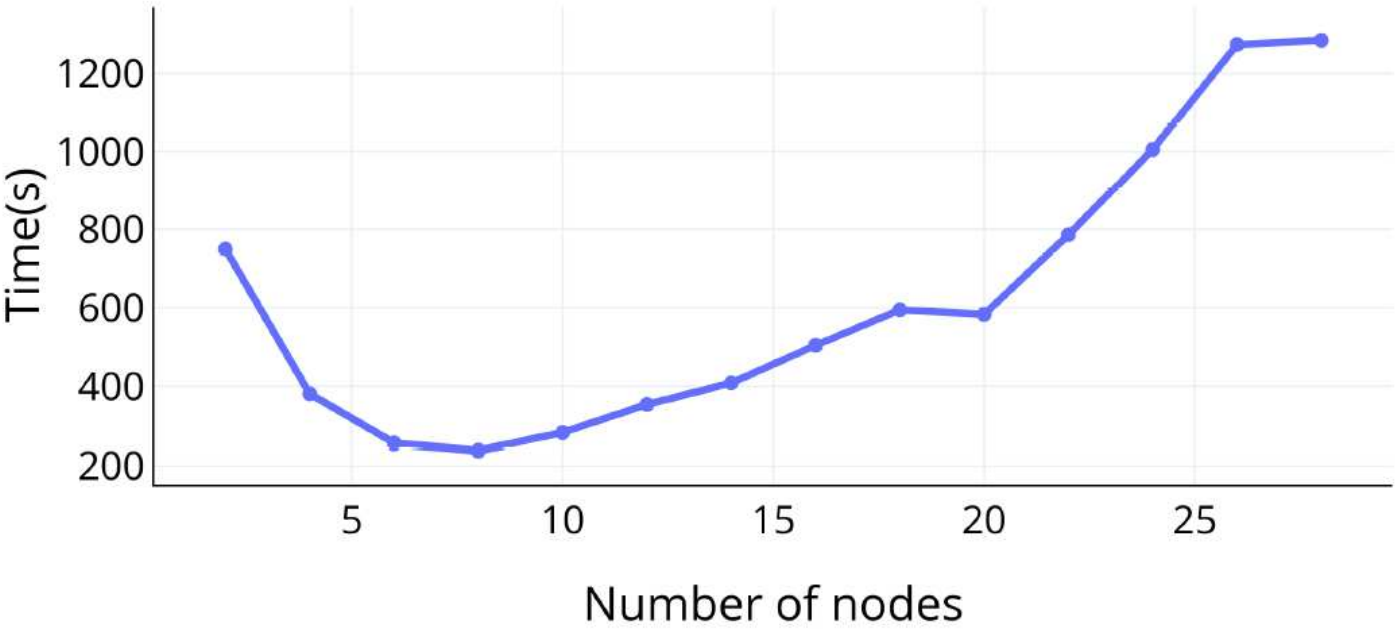

Figure 4

Scaling properties of Method 3, for the Mnist data set

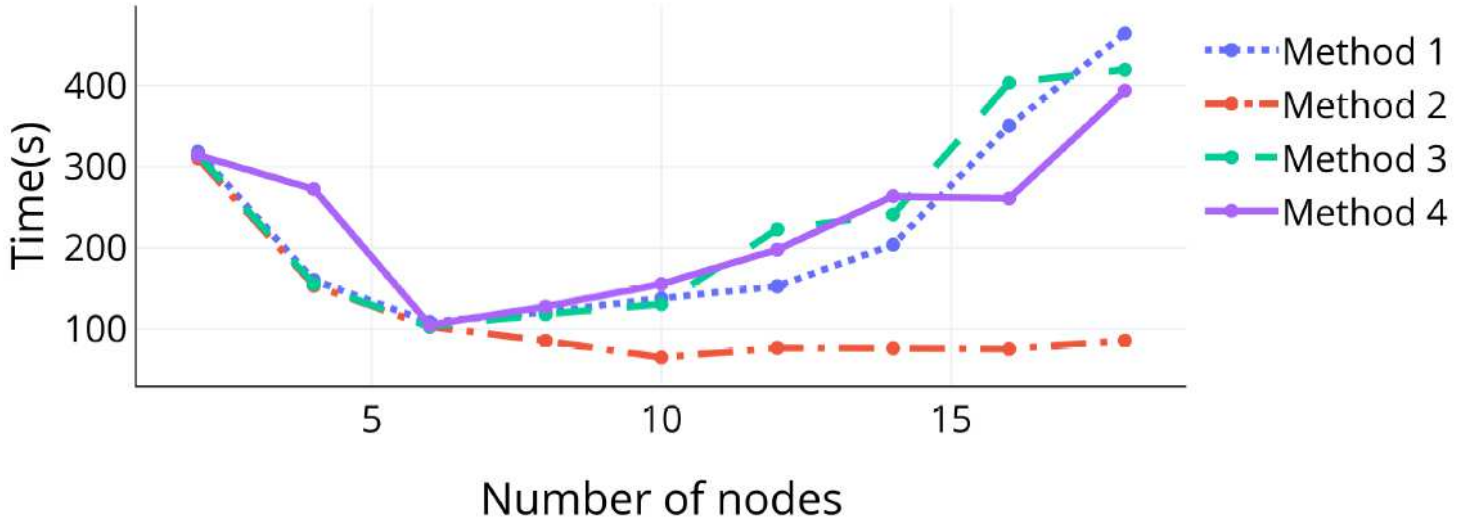

Figure 5

Execution times for the first order Methods 1-4 on CT data set

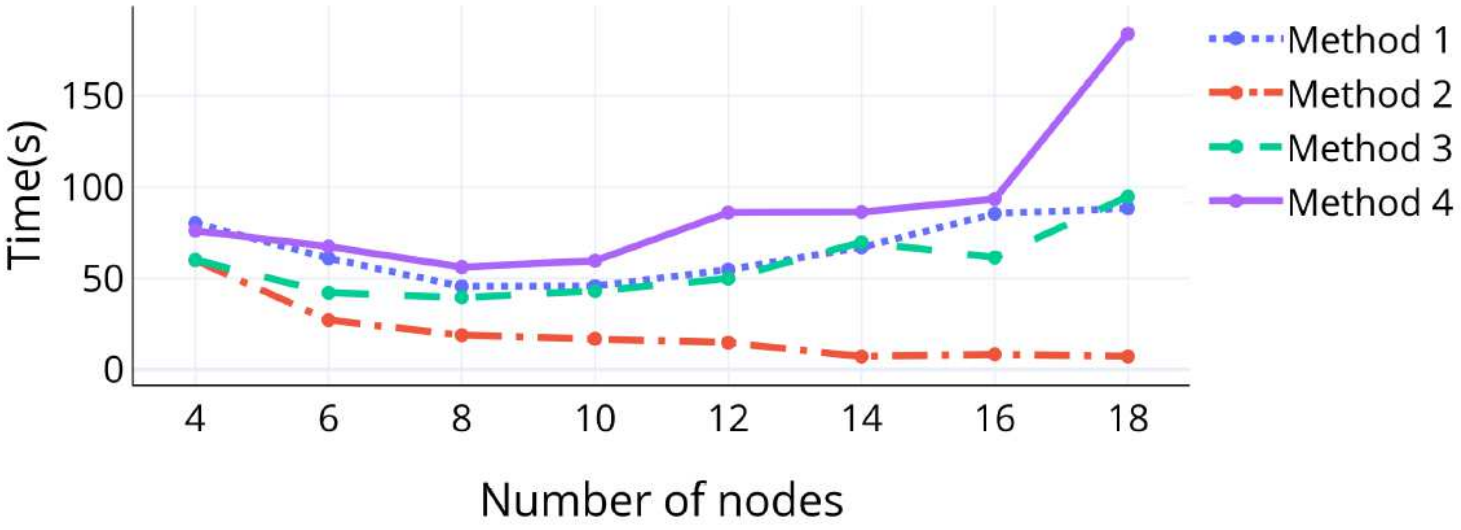

Figure 6

Execution times for the first order Methods 1-4 on Gisette data set

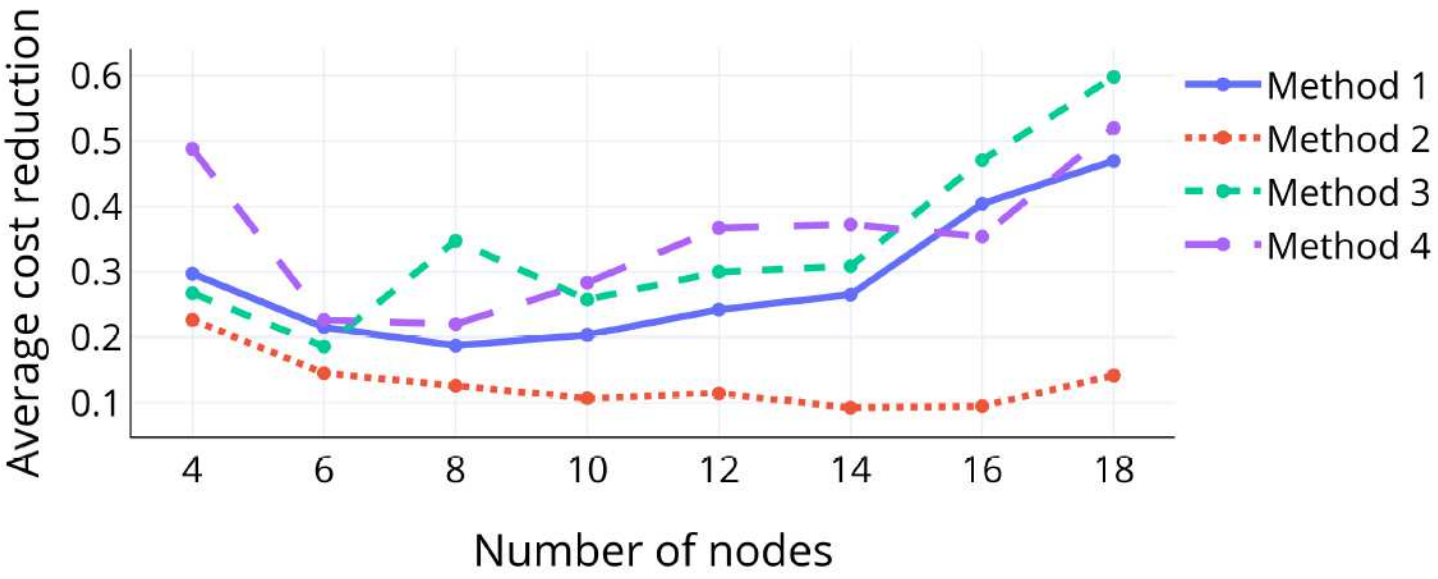

Figure 7

Average cost reduction compared to the worst relevant tested method, Methods 1, 2, 3, 4

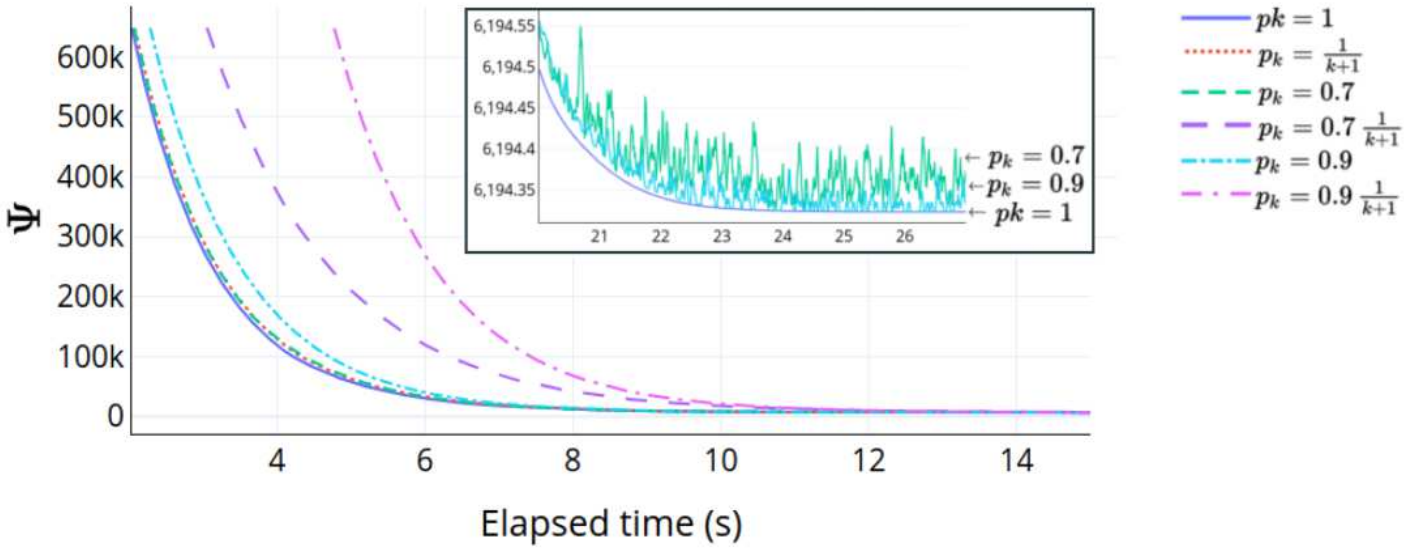

Figure 8

Comparison between using different values of pk ≤ 1, first order method, unidirectional communication, Conll data set

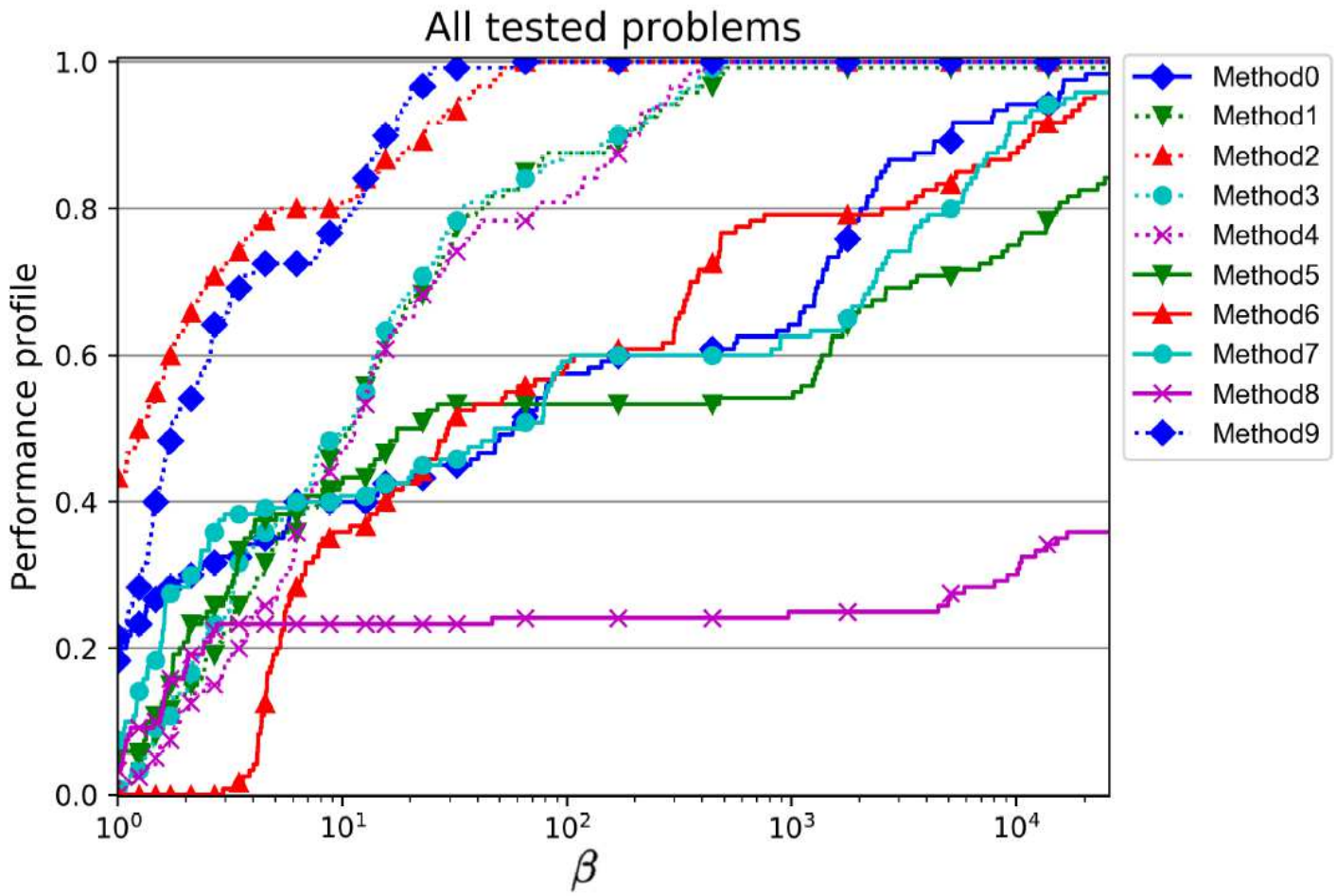

Figure 9

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

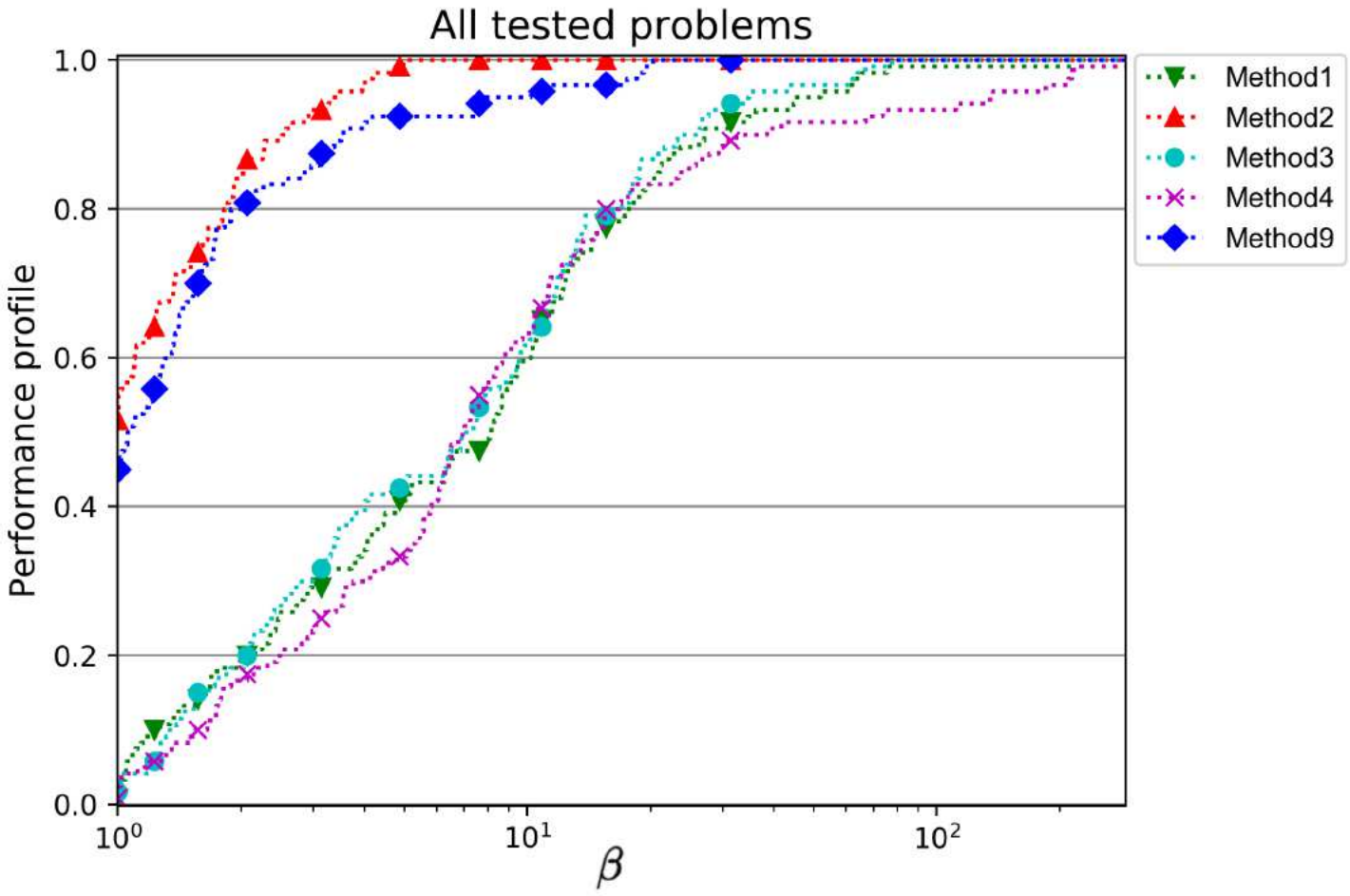

Figure 10

Performance profile for first order Methods 1, 2, 3, 4, 9, for all performed tests

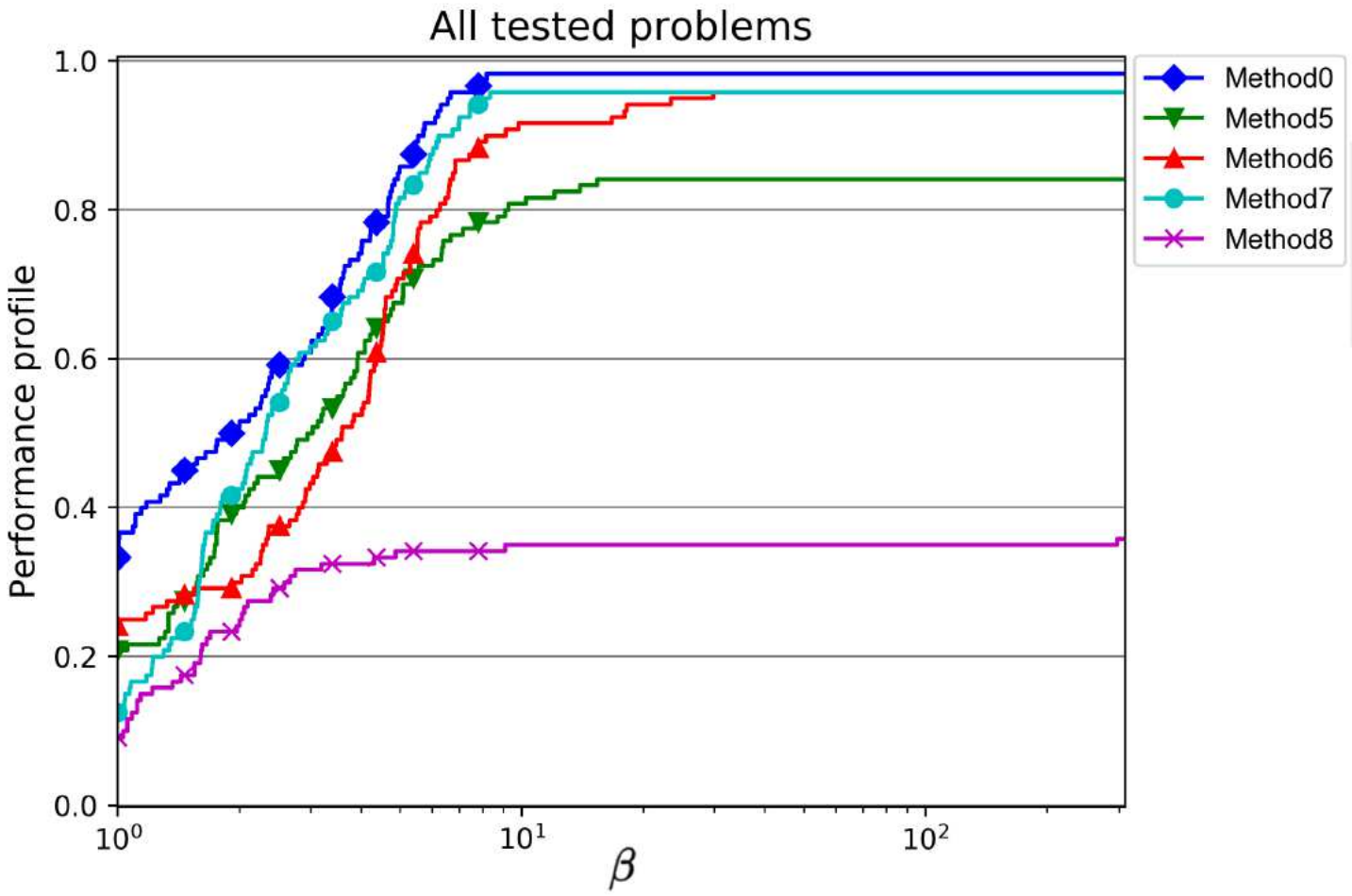

Figure 11

Performance profile for second order Methods 0, 5, 6, 7, 8, for all performed tests

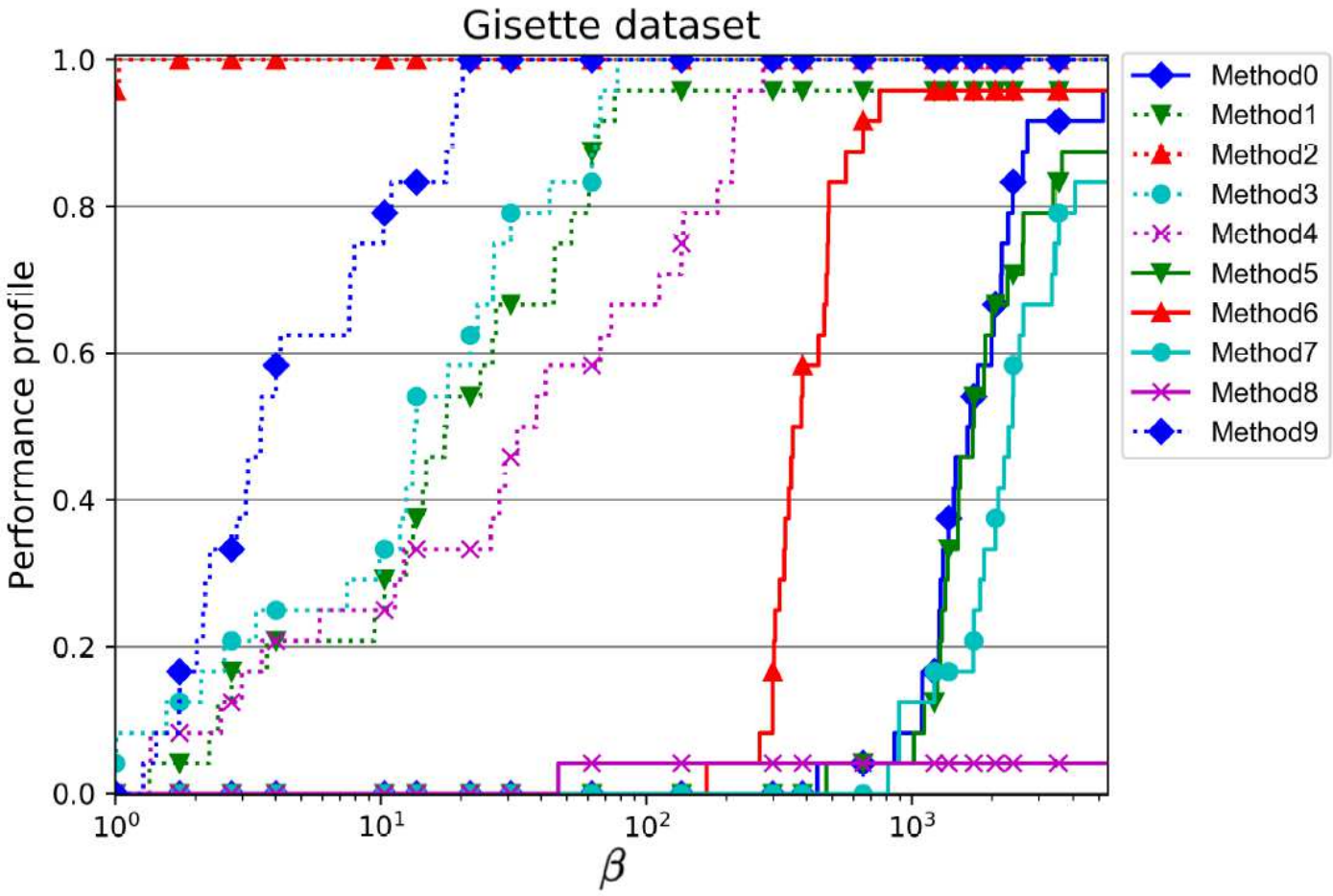

Figure 12

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

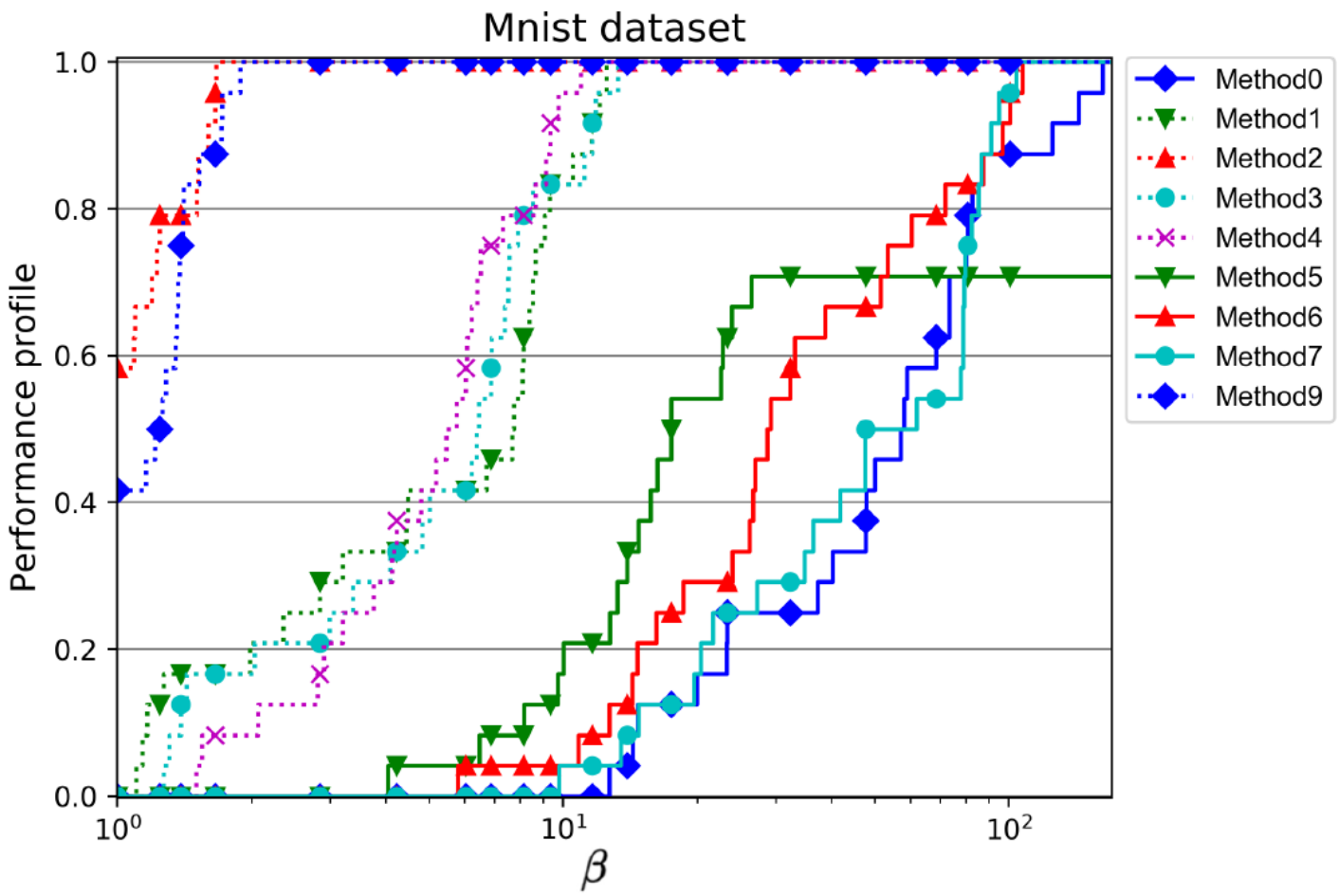

Figure 13

Performance profile for all 10 methods, for the tests performed on the Mnist data set

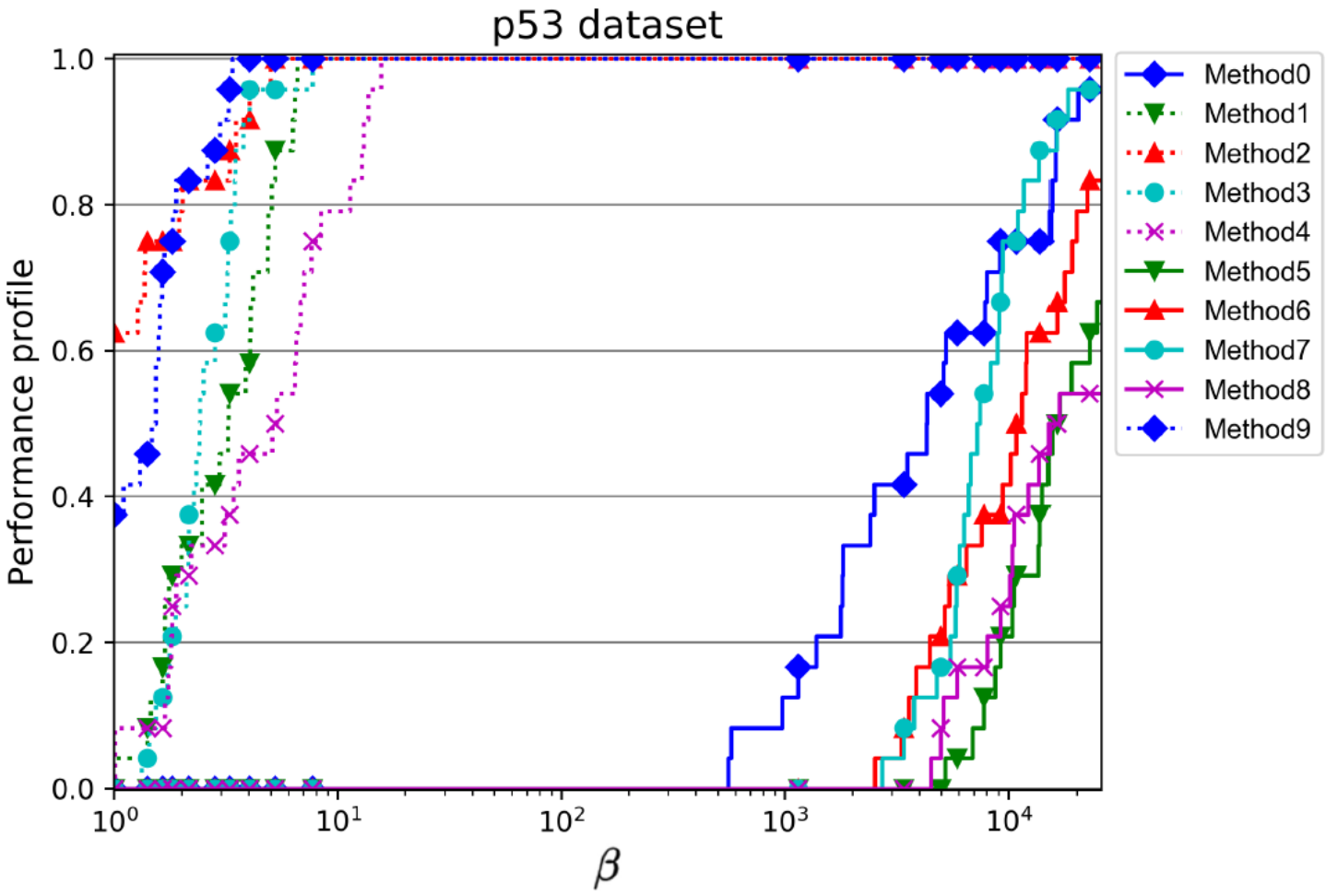

Figure 14

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

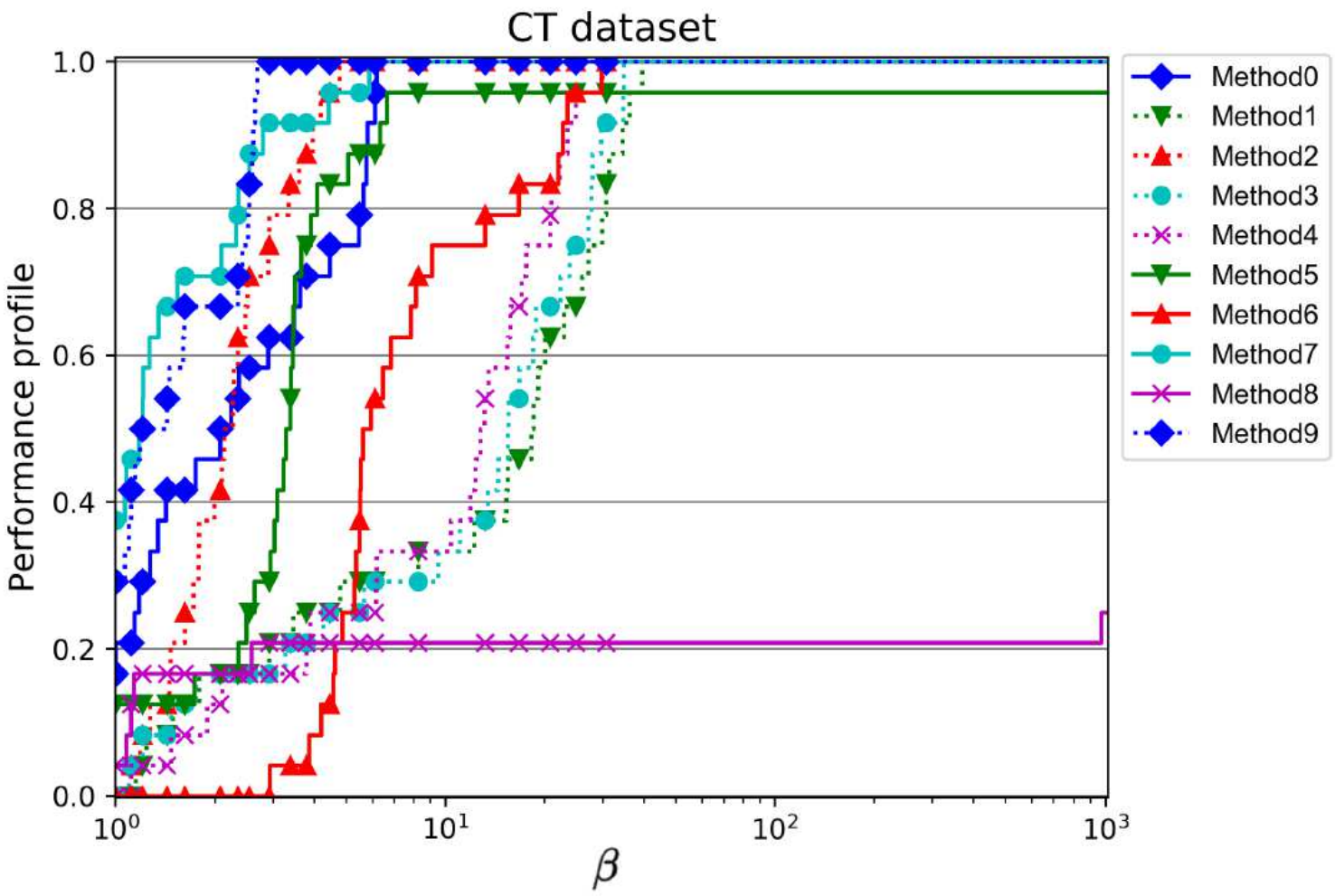

Figure 15

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

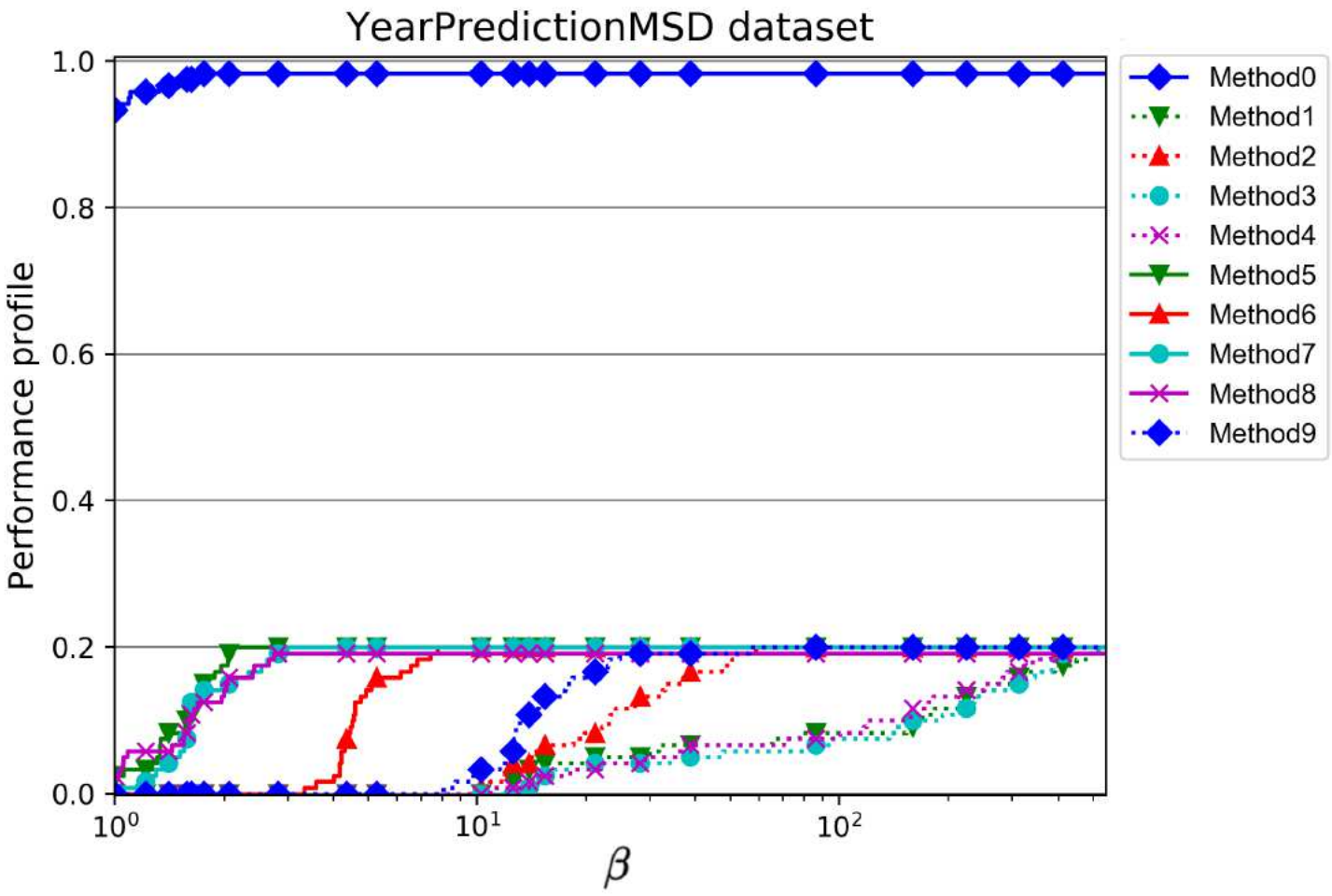

Figure 16

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

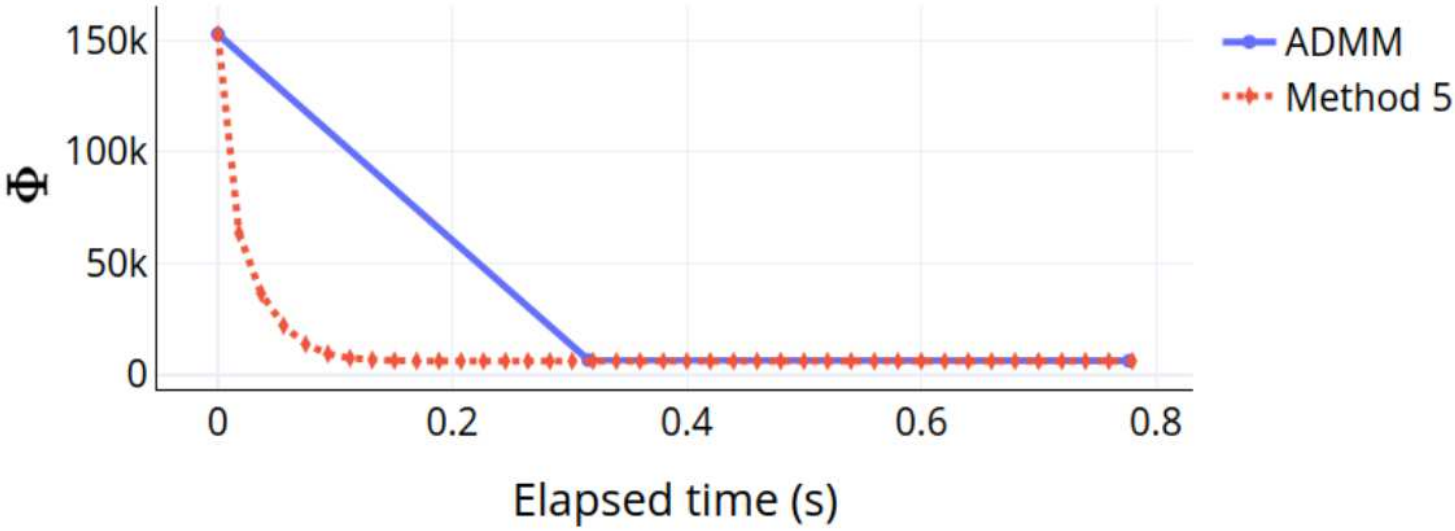

Figure 17

The comparison between ADMM and Method 5 on Conll data set