Mathematical Programming Computation (2022) 14:543–591
https://doi.org/10.1007/s12532-022-00219-z

FULL LENGTH PAPER

Limited-memory common-directions method for large-scale optimization: convergence, parallelization, and distributed optimization

Ching-pei Lee[1] · **Po-Wei Wang**[2] · **Chih-Jen Lin**[3]

Received: 25 December 2020 / Accepted: 13 February 2022 / Published online: 29 March 2022
© The Author(s) 2022

Abstract
In this paper, we present a limited-memory common-directions method for smooth optimization that interpolates between first- and second-order methods. At each iteration, a subspace of a limited dimension size is constructed using first-order information from previous iterations, and an efficient Newton method is deployed to find an approximate minimizer within this subspace. With properly selected subspace of dimension as small as two, the proposed algorithm achieves the optimal convergence rates for first-order methods while remaining a descent method, and it also possesses fast convergence speed on nonconvex problems. Since the major operations of our method are dense matrix-matrix operations, the proposed method can be efficiently parallelized in multicore environments even for sparse problems. By wisely utilizing historical information, our method is also communication-efficient in distributed optimization that uses multiple machines as the Newton steps can be calculated with little communication. Numerical study shows that our method has superior empirical performance on real-world large-scale machine learning problems.

Keywords Smooth optimization · Optimal method · First-order method · Second-order method

Mathematics Subject Classification 90C06 · 90C30 · 68W10 · 68W15 · 65K05

✉ Ching-pei Lee
leechingpei@gmail.com

Po-Wei Wang
poweiw@cs.cmu.edu

Chih-Jen Lin
cjlin@csie.ntu.edu.tw

[1] Academia Sinica, Taipei, Taiwan

[2] Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, USA

[3] Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

1 Introduction

We consider the following unconstrained smooth optimization problem.

$$\min_{\boldsymbol{w}\in\mathbf{R}^n} \quad f(\boldsymbol{w}), \tag{1}$$

where f is ρ-Lipschitz-continuously differentiable and the solution set Ω of (1) is nonempty. We propose and study a limited-memory common-directions method for (1). At each iteration, our method constructs a manifold or a subspace of a limited dimension m in which we find the update direction. In the manifold parameterized by a smooth function G, whose domain is usually of a much smaller dimension, we may apply many efficient algorithms such as the Newton method to find an update direction. That is, given the current $\boldsymbol{w}$ and the manifold parameterized by a smooth function $G : \mathbf{R}^m \to \mathbf{R}^n$, where $m \ll n$, we approximately optimize

$$\min_{\boldsymbol{t}\in\mathbf{R}^m} \quad f(\boldsymbol{w} + G(\boldsymbol{t})). \tag{2}$$

This can be done through a Newton method that iteratively solves the quadratic problem[1]

$$\min_{\boldsymbol{t}\in\mathbf{R}^m} \quad \nabla_{\boldsymbol{t}} f\left(\boldsymbol{w} + G\left(\boldsymbol{t}^{(T)}\right)\right)^\top \boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^\top \nabla^2_{\boldsymbol{t}} f\left(\boldsymbol{w} + G\left(\boldsymbol{t}^{(T)}\right)\right)\boldsymbol{t} \tag{3}$$

and conducts a line search, where $\boldsymbol{t}^{(T)}$ is the iterate obtained at the Tth Newton iteration. We call the iterations of constructing G and solving (2) the outer iterations, and the Newton steps (3) inner iterations.

Problem (2) reduces the original optimization problem to a much lower dimensional one and thus this subproblem can be solved efficiently in many cases. In particular, if we assume that G is a linear function such that

$$G(\boldsymbol{t}) = P\boldsymbol{t}, \quad \forall \boldsymbol{t} \in \mathbf{R}^m \tag{4}$$

for some $P \in \mathbf{R}^{n\times m}$, then

$$\frac{\partial G(\boldsymbol{t})}{\partial \boldsymbol{t}^\top} = P, \quad \frac{\partial^2 G(\boldsymbol{t})}{\partial \boldsymbol{t}\partial \boldsymbol{t}^\top} = 0, \quad \forall \boldsymbol{t} \in \mathbf{R}^m,$$

which reduces (3) to

$$\min_{\boldsymbol{t}\in\mathbf{R}^m} \quad \nabla f\left(\boldsymbol{w} + G\left(\boldsymbol{t}^{(T)}\right)\right)^\top P\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^\top \left(P^\top \nabla^2 f\left(\boldsymbol{w} + G\left(\boldsymbol{t}^{(T)}\right)\right) P\right)\boldsymbol{t}. \tag{5}$$

[1] We do not assume f to be twice-differentiable, but its Lipschitz-continuous differentiability ensures that it is twice-differentiable almost everywhere, and a generalized Hessian can be used [7]. We use $\nabla^2 f$ to denote an arbitrary generalized Hessian, and when f is twice-differentiable, this reduces to the Hessian.

This can be easily minimized by solving the following Newton linear system when $P^\top \nabla^2 f\left(\boldsymbol{w} + G\left(\boldsymbol{t}^{(T)}\right)\right) P$ is positive definite.

$$P^\top \nabla^2 f\left(\boldsymbol{w} + P\boldsymbol{t}^{(T)}\right) P\boldsymbol{t} = -P^\top \nabla f\left(\boldsymbol{w} + P\boldsymbol{t}^{(T)}\right). \tag{6}$$

Equivalently, each column of P is considered as a possible direction for updating $\boldsymbol{w}$, and solving the subproblem (2) finds the best linear combination of the columns of P as the update direction. This explains the nomenclature – each column of P is a "common direction" as they can be commonly used to construct G in multiple outer iterations.

Notice that $\nabla^2 f\left(\boldsymbol{w} + G\left(\boldsymbol{t}^{(T)}\right)\right) P$ can be calculated in the cost of $O(m)$ gradient evaluations of f through automatic differentiation (see, for example, [21, Chapter 8]), or in an even lower cost when the problem structure can be utilized. Thereafter, obtaining the coefficients in (6) takes $O(m^2)$ inner products of n-dimensional vectors. When m is small, the cost of calculating the gradient and $\nabla^2 f\left(\boldsymbol{w} + G\left(\boldsymbol{t}^{(T)}\right)\right) P$ dominates, thus roughly the cost per inner iteration is asymptotically equivalent to m iterations of gradient descent if m is a fixed constant not affected by the problem parameters. Moreover, in many situations when the problem possesses specific structures that can be utilized, as described in Sect. 5.2, multiple inner iterations can even be cheaper than conducting a single gradient evaluation for f, making one outer iteration of our method cost the same as one iteration of gradient descent.

In the special case in which the column space of P is a subspace of the space spanned by the gradients of f in the previous and current outer iterations, our method can be seen as a first-order method. However, our method is more general in allowing other possible directions, such as quasi-Newton or stochastic Newton ones, being included. On the other hand, as Newton's method is applied in subspaces, our method can be considered as incorporating the curvature information as well. For example, when $G(t) = \nabla f(\boldsymbol{w})t$, our method with only one inner Newton step is similar to the Borwein-Barzilai method [2] at deciding the step size of steepest descent using some spectral information; when G includes recent update directions and the corresponding gradient differences, our method is in spirit similar to quasi-Newton methods in using these directions to approximate the Newton method. Therefore, it is expected that the proposed method can enjoy advantages from both first- and second-order methods. We will show that in theory, our method enjoys low global iteration complexity just like (accelerated) first-order methods, while in practice, our method converges even faster than second-order methods.

For problems with complicated structure, evaluations of the gradient, the objective, and the Hessian may not be highly parallelizable, especially when the data in the problem are sparse such that the bottleneck becomes memory access. In this case, traditional batch methods do not enjoy parallelism as much as expected. On the other hand, in the proposed limited-memory common-directions method, the major operations of the m Hessian-vector products and the $O(m^2)$ vector-vector inner products in constructing the left-hand side matrix of (3) are by nature embarrassingly parallel at least up to m processors. This means that our method can enjoy better parallelism

than most batch methods, and the vector-vector inner products can even be conducted in a distributed manner where the P matrix is stored disjointly on multiple machines.

Another advantage of our method is in distributed optimization such that multiple machines connected through a network are simultaneously used. For this scenario, the communication cost, or equivalently the rounds of communication between machines to synchronize necessary information, is the usual bottleneck. Thus, state of the art takes fast-convergent Newton-CG methods with certain preconditioners to reduce the communication cost [13, 27, 28], but one CG iteration takes one round of communicating an n-dimensional vector. By adopting Newton directions in subspaces, the proposed method can effectively reduce the number of iterations and therefore the communication rounds required to be lower than that of existing approaches. Moreover, unlike Newton-CG approaches, with some additional computation, our method obtains the Newton direction in a subspace by communicating only $O(m^2)$ elements between machines. In essence, our method trades local computation for communication, and the former is abundant in this case as we have multiple machines. Therefore, it is much more communication-efficient and hence suitable for distributed optimization as well.

1.1 Related work and our contributions

Limited-memory methods have been extensively studied in the context of quasi-Newton methods. When the initial Hessian estimate is a multiple of the identity, it can be easily seen that quasi-Newton methods belonging to the Broyden class find the update directions from the span of the current and previous gradients. Therefore, they can be seen as a type of first-order methods that finds the linear combination coefficients of the gradients using inner products between the historical gradients and the update steps. Among quasi-Newton methods, the limited-memory BFGS (L-BFGS) method [14] is one of the most popular, thanks to its economical spatial and computational costs per iteration and superior empirical convergence.

Recently, [26] proposed the common-directions method for smooth and strongly convex empirical risk minimization (ERM) problems. Their algorithm maintains the gradients up to the current iteration and finds the update step as a linear combination of them. The combination is determined by approximately finding the best possible one through running multiple iterations of the Newton method. The key ingredient in their algorithm is to utilize the problem structure of ERM to efficiently compute the Hessian with low cost. They show that by accumulating the gradients, this method possesses both the optimal global linear convergence rate for first-order methods and local quadratic convergence from the Newton method, and empirically it outperforms both optimal first-order methods and second-order methods on small- and medium-scale problems. The major disadvantage of the common-directions method is its usage of all gradients accumulated from the first iteration on such that for a rather difficult problem that requires many iterations to solve, both the storage and the subproblem solve become expensive.

In this paper, we consider a fusion between the common-directions method and the limited-memory quasi-Newton methods to develop a limited-memory common-

directions method. In particular, instead of storing and using all gradients accumulated from the first iteration on, our method uses only information from the most recent $\hat{m}$ iterations for a pre-specified value of $\hat{m}$. To retain information from the discarded gradients for possibly better convergence, we also include the possibility of using as the common directions the momentum terms or previous update steps adopted by optimal first-order methods and limited-memory quasi-Newton methods. In comparison with our initial presentation of the limited-memory common-directions method that focused only on distributed optimization for a specific problem class [10], this paper provides a more general treatment for smooth optimization with extended theoretical analysis for optimal convergence rates, broader applicability, improved algorithms, and extensive experiments. In particular, our Sect. 5.2 is a generalization of the algorithmic description in [10], Theorems 2 and 5 are adapted from [10] to our more general setting, and Theorem 3 improves upon the result in the same work, but other parts are newly developed.

Through this limited-memory setting, we obtain controllable spatial and computational per-iteration cost, extend applicability from ERM to general optimization problems with the help of automatic differentiation, and better parallelize the computation. We develop convergence results using techniques different from that in [26] because we no longer include all previous gradients in our search space. We show that the optimal linear convergence rate for first-order methods on strongly convex problems is still attainable even when the number of common directions at each iteration is as small as two. In addition, we also cover the case not shown in [26] to prove that optimal convergence rate of $O(1/k^2)$ for first-order methods on general convex problems can also be obtained with two properly selected common directions (k is the outer iteration counter in our method). Unlike other optimal methods that are non-monotone, our method is a strictly descent one. Moreover, other optimal methods only possess R-linear convergence on strongly convex problems, but our method also achieves global Q-linear convergence (with a different rate). Another contribution of this work in theory is showing that for a broad choices of the common directions, even if (2) is solved as coarse as by only one Newton iteration, global sublinear convergence rates of $O(1/k)$ on both convex and nonconvex (to stationarity) problems, and global Q-linear convergence on problems satisfying the Polyak-Łojasiewicz condition can be ensured.

We also discuss that our algorithm is also suitable for multicore parallelization and distributed optimization to practically solve large-scale problems with high efficiency, and show through numerical results that the empirical behavior of the proposed algorithm is indeed suitable for these scenarios and can outperform state of the art in multicore parallel optimization and distributed optimization.

Empirical studies also show that our method outperforms optimal first-order methods on a single-core setting, and state of the art methods in multicore and distributed environments, and hence we have included the distributed version of the proposed algorithm in the open-source package MPI-LIBLINEAR.[2]

[2] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear/.

1.2 Notations and assumptions

We denote $f^* := \min_{\boldsymbol{w}} f(\boldsymbol{w})$. Given any $\epsilon \geq 0$, we say that a point $\boldsymbol{w}$ is an ϵ-accurate solution for (1) if $f(\boldsymbol{w}) \leq f^* + \epsilon$. For a given set of vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_t$, $\texttt{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_t)$ denotes the subspace spanned by them. That is,

$$\texttt{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_t) := \left\{ \sum_{i=1}^{t} \beta_i \boldsymbol{a}_i \,\middle|\, \beta_i \in \Re, i = 1, \ldots, t \right\}.$$

For any function G, $\texttt{Im}(G)$ and $\texttt{dom}(G)$ denote its image and effective domain, respectively. When not specified otherwise, $\|\cdot\|$ signifies $\|\cdot\|_2$. Given two symmetric matrices A and B of the same dimension, $A \succeq B$ means that $A - B$ is positive semidefinite and $A \succ B$ means $A - B$ is positive definite. I denotes the identity matrix, and $\sigma_{\min}(A)$ the smallest eigenvalue of A. We denote the kth iterate by $\boldsymbol{w}_k$ for all $k \geq 0$.

The following is assumed throughout this work.

Assumption 1 The objective f in (1) is ρ-Lipschitz-continuously differentiable for some $\rho > 0$. Moreover, the solution set Ω of (1) is non-empty.

We note that restricting the domain in (1) to $\mathbf{R}^n$ is just for the ease of description, and our algorithm and analysis apply directly to any Euclidean spaces.

1.3 Organization

This work is organized as follows. Section 2 describes and analyzes a version of our algorithmic framework that achieves the optimal convergence rates for first-order methods with a carefully selected G. A more general version is then given in Sect. 3 with its convergence rates analyzed in Sect. 4. Discussion in Sect. 5.1 studies the choice of the common directions for improving the empirical convergence and how to utilize some special problem structures to make our method highly efficient in Sect. 5.2. Numerical results are presented in Sect. 6 to examine the empirical performance of our method. We further apply the proposed algorithm to parallel and distributed optimization in Sects. 7 and 8, respectively. Section 9 provides some concluding remarks. All proofs are in the "Appendix".

2 Limited-memory common-directions method with optimal convergence rates

We start with a case in which our method can be viewed as a first-order method, in the sense that at the kth outer iteration,

$$\boldsymbol{w}_k \in \boldsymbol{w}_0 + \texttt{span}(\nabla f(\boldsymbol{w}_0), \ldots, \nabla f(\boldsymbol{w}_{k-1}))$$

for all $k \geq 0$. It is known that the optimal iteration complexity for first-order methods to reach an ϵ-accurate solution under Assumption 1 is $O(1/\sqrt{\epsilon})$ when f is convex (or

equivalently the best convergence rate is $O(1/k^2)$), and $O(\sqrt{\rho/\sigma}\log(1/\epsilon))$, from an R-linear convergence rate, when f is σ-strongly convex with $\sigma > 0$ [17, 19]. We will show that for a properly chosen G with m as small as two, our method achieves such optimal iteration complexities if (2) is approximately solved to enough accuracy.

We describe our method as a meta algorithm in Algorithm 1, such that only outer iterations are considered while skipping the details of solving (2). There are many possible algorithms to efficiently optimize (2) when G is linear with a small dimension, and we postpone this discussion to later sections.

The choice of G can be quite arbitrary as long as it is linear, and for achieving the optimal convergence speed of first-order methods on σ-strongly convex problems for some $\sigma \geq 0$ ($\sigma = 0$ indicates that f is only convex), we only require that at the kth outer iteration, $\mathtt{Im}(G)$ contains $\mathtt{span}\,(\nabla f(\boldsymbol{w}_k), \boldsymbol{v}_{k+1} - \boldsymbol{w}_k)$, where $\boldsymbol{v}_{k+1}$ is defined by

$$\begin{cases} \boldsymbol{v}_0 & := \boldsymbol{w}_0, \\ \boldsymbol{v}_{k+1} & := \frac{1}{\gamma_{k+1}}\left((1-\alpha_k)\,\gamma_k \boldsymbol{v}_k + \alpha_k \sigma \boldsymbol{w}_k - \alpha_k \nabla f\,(\boldsymbol{w}_k)\right), \forall k \geq 0, \end{cases} \tag{7}$$

with

$$\gamma_0 = \begin{cases} \rho & \text{if } \sigma = 0, \\ \sigma & \text{else,} \end{cases} \qquad \gamma_{k+1} := (1-\alpha_k)\,\gamma_k + \alpha_k \sigma, \quad \forall k \geq 0, \tag{8}$$

and α_k being the positive solution to

$$\rho \alpha_k^2 = (1-\alpha_k)\,\gamma_k + \alpha_k \sigma. \tag{9}$$

When $\sigma > 0$, Eqs. (8) and (9) imply

$$\gamma_k \equiv \sigma, \quad \alpha_k \equiv \alpha := \sqrt{\frac{\sigma}{\rho}}, \quad \forall k \geq 0. \tag{10}$$

When $\sigma = 0$, (8) and (9) in combination shows that

$$\gamma_{k+1} = \rho \alpha_k^2, \quad \forall k \geq 0. \tag{11}$$

Therefore from (9),

$$\rho \alpha_{k+1}^2 = \gamma_{k+1}\,(1-\alpha_{k+1}) = \rho \alpha_k^2 (1-\alpha_{k+1}), \quad \forall k \geq 0, \tag{12}$$

and from the quadratic formula applied to (12) for $k > 0$ and to (8)–(9) for $k = 0$, we have

$$\alpha_0 = \frac{-1+\sqrt{5}}{2}, \quad \alpha_{k+1} = \frac{-\alpha_k^2 + \alpha_k\sqrt{\alpha_k^2+4}}{2}, \quad \forall k \geq 0. \tag{13}$$

By induction, it is clear that $\alpha_k > 0$ for all k, so (10) and (11) imply that $\gamma_{k+1} > 0$ for all k. We can therefore see that no matter $\sigma > 0$ or $\sigma = 0$, $\boldsymbol{v}_k$ in (7) is always well-defined.

Using simple induction, we can see that

$$\boldsymbol{w}_k, \boldsymbol{v}_{k+1} \in \boldsymbol{w}_0 + \mathtt{span}\left(\nabla f(\boldsymbol{w}_0), \ldots, \nabla f(\boldsymbol{w}_k)\right),$$

so after including the span of $\boldsymbol{v}_{k+1} - \boldsymbol{w}_k$, our method can still be considered as a first-order method. Notice that at the beginning of the kth iteration, $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are already known, so we can obtain $\boldsymbol{v}_{k+1}$ before starting the kth iteration of the algorithm. For the case in which f is nonconvex, we are able to show that $\min_{0\le T\le k} \|\nabla f(\boldsymbol{w}_T)\|^2$ converges to 0 at a rate of $o(1/k)$, which is the same speed as gradient descent and the accelerated gradient method for nonconvex problems, as long as $\mathtt{Im}(G)$ includes a vector whose direction does not deviate from the reversed gradient direction too much. Interestingly, unlike the result for the convex cases that requires sufficient accurate subproblem solutions for (2), this rate for stationarity can be achieved even if the subproblem is solved very roughly, as we shall see later in Theorem 3 in Sect. 4.

Algorithm 1: Limited-memory Common-directions Method

Given $\boldsymbol{w}_0 \in \mathbf{R}^n$ **for** *k=0,1,...* **do**
  Pick a positive integer m and linear function $G : \mathbf{R}^m \to \mathbf{R}^n$
  Approximately solve

$$\boldsymbol{t}_k \approx \arg\min_{\boldsymbol{t}\in\mathbf{R}^m} f(\boldsymbol{w}_k + G(\boldsymbol{t})) \tag{14}$$

  $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + G(\boldsymbol{t}_k)$
end

We prove below the optimality of Algorithm 1 in terms of the convergence speed.

Theorem 1 *Consider (1) with f σ-strongly convex for some $\sigma \ge 0$ ($\sigma = 0$ implies that f is only convex). For any initial point $\boldsymbol{w}_0$, the following results hold.*

1. *If $\sigma > 0$ and the subproblem solution for (2) at the kth iteration generates the next iterate $\boldsymbol{w}_{k+1}$ that satisfies the following conditions for all $k \ge 0$,*

$$\begin{cases} \nabla f\left(\boldsymbol{w}_{k+1}\right)^\top \left(\boldsymbol{v}_{k+1} - \boldsymbol{w}_{k+1}\right) + \frac{\sigma}{2}\left\|\boldsymbol{v}_{k+1} - \boldsymbol{w}_{k+1}\right\|^2 \ge 0, \\ f\left(\boldsymbol{w}_{k+1}\right) \le f\left(\boldsymbol{w}_k\right) - \frac{1}{2\rho}\left\|\nabla f\left(\boldsymbol{w}_k\right)\right\|^2, \end{cases} \tag{15}$$

then Algorithm 1 generates objective values R-linearly convergent to the optimum:

$$f\left(\boldsymbol{w}_k\right) - f^* \le \left(1 - \sqrt{\frac{\sigma}{\rho}}\right)^k \left(\frac{\sigma}{2}\left\|\boldsymbol{w}_0 - \boldsymbol{w}^*\right\|^2 + f\left(\boldsymbol{w}_0\right) - f^*\right), \tag{16}$$

and it takes $O(\sqrt{\rho/\sigma}\log(1/\epsilon))$ iterations to obtain an ϵ-accurate solution.

2. *If $\sigma = 0$ and there exists a sequence $\{\psi_i\}$ (not necessarily all nonnegative) such that for all $k \geq 0$,*

$$\begin{cases} \nabla f\left(\boldsymbol{w}_{k+1}\right)^{\top}\left(\boldsymbol{v}_{k+1}-\boldsymbol{w}_{k+1}\right)+\psi_{k+1} \geq 0, \\ f\left(\boldsymbol{w}_{k+1}\right) \leq f\left(\boldsymbol{w}_k\right)-\frac{1}{2\rho}\left\|\nabla f\left(\boldsymbol{w}_k\right)\right\|^2, \end{cases} \tag{17}$$

then the iterates generated by Algorithm 1 satisfy

$$f\left(\boldsymbol{w}_k\right)-f^* \leq \frac{4}{(k+1)^2}\left(\frac{\rho}{2}\left\|\boldsymbol{w}_0-\boldsymbol{w}^*\right\|^2+f\left(\boldsymbol{w}_0\right)-f^*+\sum_{i=1}^{k-1}\frac{\psi_i}{\alpha_i}\right), \tag{18}$$

for all $k \geq 0$ for any $\boldsymbol{w}^*$ in the solution set. Therefore, if there exists a constant $A_\infty < \infty$ such that

$$A_k := \sum_{i=1}^{k}\frac{\psi_i}{\alpha_i} \leq A_\infty, \quad \forall k > 0, \tag{19}$$

Algorithm 1 takes $O(1/\sqrt{\epsilon})$ iterations to get an ϵ-accurate solution.

In Theorem 1, if ψ_k converges to zero at a rate of $O(\alpha_k k^{-(1+\delta)})$ for any $\delta > 0$, (19) is satisfied. (We can simply set $\psi_k = \alpha_k/(k^{1+\delta})$ for any $\delta > 0$.) When $\sigma = 0$, one can also replace α_k defined by (9) with some simpler choices such as $\alpha_k = (q-1)/(k+q-1)$ with $q > 2$ proposed by [25]. In this case, the optimal $O(k^{-2})$ rate is still obtained, but the convergence rate requirement of ψ_k can be simplified to $O(k^{-(2+\delta)})$ for any $\delta > 0$. The proof is essentially the same by noticing that in this case, instead of the equality in (12), we have

$$\rho\alpha_k^2 \geq (1-\alpha_k)\,\rho\alpha_{k-1}^2.$$

We then show that when G is selected properly, (15) and (17) can be met by solving (2) approximately.

Proposition 1 *Assume f is σ-strongly convex for some $\sigma > 0$. Consider (2) at the kth iteration with $\boldsymbol{w} = \boldsymbol{w}_k$ and let*

$$\tilde{\boldsymbol{v}}_{k+1} := \boldsymbol{v}_{k+1} - \boldsymbol{w}_k. \tag{20}$$

If $\mathtt{span}\left(\nabla f\left(\boldsymbol{w}_k\right), \tilde{\boldsymbol{v}}_{k+1}\right) \subseteq \mathtt{Im}(G)$ and G is a linear function, then either

$$\begin{cases} \nabla f\left(\boldsymbol{w}_k+G\left(\boldsymbol{t}\right)\right)^{\top}\left(\tilde{\boldsymbol{v}}_{k+1}-G\left(\boldsymbol{t}\right)\right)+\frac{\sigma}{2}\left\|\tilde{\boldsymbol{v}}_{k+1}-G\left(\boldsymbol{t}\right)\right\|^2 \geq 0, & \text{(21a)} \\ f\left(\boldsymbol{w}_k+G\left(\boldsymbol{t}\right)\right) \leq f\left(\boldsymbol{w}_k\right)-\frac{1}{2\rho}\left\|\nabla f\left(\boldsymbol{w}_k\right)\right\|^2 & \text{(21b)} \end{cases}$$

holds for all $\boldsymbol{t}$ within some neighborhood of the solution set for (2); or either $G(\boldsymbol{t}^*) = \tilde{\boldsymbol{v}}_{k+1}$ or $G(\boldsymbol{t}^*) = -\nabla f(\boldsymbol{w}_k)/\rho$ is satisfied by all optimal solutions $\boldsymbol{t}^*$.

Proposition 2 *Assume f is convex and $\sigma = 0$ in (7)–(9). Consider (2) at the kth iteration with $\boldsymbol{w} = \boldsymbol{w}_k$. If $\mathtt{span}(\nabla f(\boldsymbol{w}_k), \tilde{\boldsymbol{v}}_{k+1}) \subseteq \mathtt{Im}(G)$ and G is a linear function, then given any $\psi_{k+1} > 0$, either*

$$\begin{cases} \nabla f\left(\boldsymbol{w}_k + G\left(\boldsymbol{t}\right)\right)^\top \left(\tilde{\boldsymbol{v}}_{k+1} - G\left(\boldsymbol{t}\right)\right) + \psi_{k+1} \geq 0, & (22a) \\ f\left(\boldsymbol{w}_k + G\left(\boldsymbol{t}\right)\right) \leq f\left(\boldsymbol{w}_k\right) - \frac{1}{2\rho} \left\|\nabla f\left(\boldsymbol{w}_k\right)\right\|^2 & (22b) \end{cases}$$

holds for all $\boldsymbol{t}$ within some neighborhood of the solution set for (2); or there is an optimal $\boldsymbol{t}^*$ for (2) satisfies $G(\boldsymbol{t}^*) = -\nabla f(\boldsymbol{w}_k)/\rho$ and this $\boldsymbol{t}^*$ satisfies (22).

Notice that (21) and (22) respectively imply (15) and (17) in the next iteration. The above propositions therefore imply that either any approximate solution of (2) close enough to the solution set or some easily calculable vectors satisfy the required conditions for ensuring optimal convergence rates in Theorem 1. Thus, we can apply any convergent iterative subproblem solver to (2) and get the condition (21) or (22) satisfied within finite iterations. More specifically, when f is strongly convex, the minimizer is unique, and thus any algorithm ensuring the convergence of the objective to the minimum will generate iterates that eventually reach the neighborhood satisfying (21). When f is convex, any algorithm that produces an iterate sequence that converges to a minimizer, such as gradient descent [4, 11], accelerated gradient [with suitable parameters,1, 5], randomized coordinate descent [with probability one,23], will be able to reach the neighborhood that satisfies (22) in finite iterations. In fact, in our experiment in Sect. 6 for strongly convex f, we observe that the condition (21) is always satisfied after one Newton step.

Including $\boldsymbol{v}_{k+1} - \boldsymbol{w}_k$ in $\mathtt{Im}(G)$ as suggested by Propositions 1 and 2 requires prior knowledge of the parameters σ and ρ. When these values are unknown, one can use the line search techniques in [20] to obtain similar rates. We omit details for this case as the analysis is very similar to that in Theorem 1.

We can also obtain a $o(1/k)$ sublinear convergence rate for $\|\nabla f(\boldsymbol{w}_k)\|^2$ even if f is nonconvex, see Theorem 3 in Sect. 4.

3 Practical limited-memory common-directions method

In this section, we discuss an efficient solver for the subproblem (2) with G in the form (4). More specifically, we show how to apply a line-search Newton method (5) on the subspace selected. For the subproblem, G can be decided freely and is not limited to that suggested by Propositions 1 and 2.

We describe how one iteration of the Newton method for the subproblem is done, and if one wants to run multiple inner iterations, it is just a repetition of this procedure. Because $\mathtt{dom}(G)$ is usually of a very low dimension, we use a full Newton step without truncation. This means that given the current iterate $\boldsymbol{w}$, we obtain an update direction $\boldsymbol{t}$ for the subproblem (2) through solving the following m by m linear system constructed from (6) with $\boldsymbol{t}^{(0)} = \boldsymbol{0}$.

$$P^\top \nabla^2 f\left(\boldsymbol{w}\right) P\boldsymbol{t} = -P^\top \nabla f\left(\boldsymbol{w}\right). \tag{23}$$

Computation of the coefficients in this linear system is easy. First, given P, we can use automatic differentiation to compute the matrix $\nabla^2 f(\boldsymbol{w})\, P$. Then the computation of both $P^\top \left(\nabla^2 f(\boldsymbol{w})\, P\right)$ and $P^\top \nabla f(\boldsymbol{w})$ is straightforward. When the problem structure is known, we may also utilize it to get an even lower cost in constructing the linear system, as we will discuss in Sect. 5.2. The final update direction for $\boldsymbol{w}$ is then constructed as $\boldsymbol{p} := G(\boldsymbol{t}) = P\boldsymbol{t}$.

Two pitfalls of directly using the $\boldsymbol{p}$ obtained above require extra care. The first one is that when the matrix on the left-hand side of (23) is not positive definite, it is possible that the resultant $\boldsymbol{t}$ is not a descent direction for (2) and therefore $P\boldsymbol{t}$ may be a nondescent direction for f. To take care of this problem, we add a sufficiently large multiple of the identity to the matrix on the left-hand side of (23) whenever the smallest eigenvalue of it is smaller than a given threshold $\tau > 0$.

The second pitfall is that the full Newton's step does not always ensure sufficient objective value decrease. Therefore, to ensure convergence, we conduct a line search procedure to find a suitable step size θ and update the iterate $\boldsymbol{w}$ by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \theta \boldsymbol{p}.$$

Because we want to use the unit step size whenever possible in second-order methods, we use a simple backtracking line search procedure such that given parameters $\beta, c_1 \in (0, 1)$, we iteratively try $i = 0, 1, \ldots$ until

$$f\left(\boldsymbol{w} + \beta^i \boldsymbol{p}\right) \leq f(\boldsymbol{w}) + c_1 \beta^i \nabla f(\boldsymbol{w})^\top \boldsymbol{p}, \tag{24}$$

and let $\theta = \beta^i$. The overall algorithm is summarized in Algorithm 2.

As a side note, there are many possible options for approximately solving (2) and the described method of Newton steps with line search is just one of them. For example, we can replace the matrix on the left-hand side of (23) with any positive definite matrix and the convergence is still guaranteed, as we will see in Sect. 4. We will also discuss in Sect. 5.2 a class of problems that we can evaluate $\nabla f(G(\boldsymbol{t}))$ and $\nabla^2 f(G(\boldsymbol{t}))$ cheaply for multiple $\boldsymbol{t}$ with a fixed G, in which case running multiple inner Newton iterations can be much cheaper than updating G, so that running multiple Newton iterations and running a single Newton iteration have almost the same cost per outer iteration.

3.1 Cost per iteration

Let the cost of calculating the gradient $\nabla f(\boldsymbol{w})$ be denoted by $\mathrm{T}_{\mathrm{grad}}$. By using automatic differentiation, it takes at most $O(m\mathrm{T}_{\mathrm{grad}})$ to evaluate $\nabla^2 f(\boldsymbol{w}_k)\, P_k$ and the total cost of forming the coefficients on the left-hand side is therefore $O(m\mathrm{T}_{\mathrm{grad}} + m^2 n)$, where the second term is for the matrix-matrix product. The cost for computing the right-hand side of (23) is simply $O(mn)$ for the matrix-vector product. The cost of computing the smallest eigenvalue of $P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k$ is $O(m^3)$, which is the same as inverting H_k. This cost is usually negligible in comparison to the $O(m^2 n)$ cost of the matrix-matrix product. The cost of the function value evaluation in the line search is no greater than $O(\mathrm{T}_{\mathrm{grad}})$ and usually much smaller. In addition, we will see in Sect. 4 that the

Algorithm 2: Practical Limited-memory common-directions method

Given $\boldsymbol{w}_0$, $m > 0$, $M_2 > 0$, and $\beta, c_1 \in (0, 1)$. Compute $\nabla f(\boldsymbol{w}_0)$ and pick an initial P_0 that has no more than m columns
for *k=0,1,...* **do**
  Compute $\nabla^2 f(\boldsymbol{w}_k) P_k$
  Compute $P_k^\top (\nabla^2 f(\boldsymbol{w}_k) P_k)$ and $P_k^\top \nabla f(\boldsymbol{w}_k)$
  if $\sigma_{\min}(P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k) < M_2$ **then**
    $H_k \leftarrow (P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k) + (M_2 - \sigma_{\min}(P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k)) I$
  else
    $H_k \leftarrow P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k$
  end
  $\boldsymbol{t}_k \leftarrow -H_k^{-1} P_k^\top \nabla f(\boldsymbol{w}_k)$
  $\boldsymbol{p}_k \leftarrow P_k \boldsymbol{t}_k$
  $\Delta_k \leftarrow \nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k$
  for *i=0,1,...* **do**
    $\theta_k \leftarrow \beta^i$
    if $f(\boldsymbol{w}_k + \theta_k \boldsymbol{p}_k) \le f(\boldsymbol{w}_k) + c_1 \theta_k \Delta_k$ **then**
      Break
    end
  end
  $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \theta_k \boldsymbol{p}_k$
  Compute $\nabla f(\boldsymbol{w}_{k+1})$
  Pick some P_{k+1} that has no more than m columns
end

number of backtracking steps is bounded by a constant, and in practice $\theta_k = 1$ usually produces sufficient function decrease. Therefore, the cost per iteration of Algorithm 2 is $O(m\mathrm{T}_{\mathrm{grad}} + m^2 n + m^3)$.

4 Convergence analysis

We discuss the convergence speed of Algorithm 2. We separately discuss the three cases in which f is nonconvex, f is convex, and f satisfies the Polyak-Łojasiewicz condition [9, 15, 22], respectively.

Instead of the specific matrix H_k considered in Algorithm 2, we prove convergence rates for a more general setting in finding the update direction $\boldsymbol{p}_k$. In particular, we consider solving

$$\begin{aligned} \min_{\boldsymbol{t}_k} \quad & \nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k + \frac{1}{2} \boldsymbol{p}_k^\top \hat{H}_k \boldsymbol{p}_k \\ \text{subject to} \quad & \boldsymbol{p}_k = P_k \boldsymbol{t}_k, \end{aligned} \tag{25}$$

where $\hat{H}_k$ is a given symmetric positive definite matrix such that there exist $M_1 \ge M_2 > 0$ satisfying

$$M_1 I \succeq H_k \succeq M_2 I, \forall k. \tag{26}$$

We denote the columns of P_k by

$$P_k = [\boldsymbol{q}_1^k, \ldots, \boldsymbol{q}_{m_k}^k] \in \mathbf{R}^{n \times m_k}, \tag{27}$$

and m_k can change with k without any restriction except for $m_k > 0$ for all k.

For the ease of description, we assume without loss of generality that P_k is orthonormal, which can always be achieved through the Gram-Schmidt process with cost $O(m_k^2 n)$, which is no larger than the cost discussed in Sect. 3.1. In this case, Algorithm 2 is a special case of the framework described here because

$$P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k + aI = P_k^\top (\nabla^2 f(\boldsymbol{w}_k) + aI) P_k$$

shows that the lower bound in (26) is satisfied automatically, and since ∇f is ρ-Lipschitz continuous, $-\rho I \preceq \nabla^2 f \preceq \rho I$ and hence $M_1 \leq 2\rho + M_2$.

The following theorems show the finite termination of the backtracking and present the iteration complexity of our method with only minimal restrictions on the choice of the vectors in P_k.

Theorem 2 *Consider* (25) *for* $f(\boldsymbol{w})$ *satisfying Assumption* 1. *If at iteration* k,

$$\frac{\left|\nabla f(\boldsymbol{w}_k)^\top \boldsymbol{q}_j^k\right|}{\|\nabla f(\boldsymbol{w}_k)\| \|\boldsymbol{q}_j^k\|} \geq \delta > 0, \text{ for some } \boldsymbol{q}_j^k \text{ in } (27), \tag{28}$$

and the update direction $\boldsymbol{p}_k$ *is the same as that defined in* (25) *with* H_k *satisfying* (26), *then the backtracking line search for* (24) *with any given* $\beta, c_1 \in (0, 1)$ *terminates in finite steps, and the final step size satisfies*

$$\theta_k \geq \bar{\theta} := \min\left(1, \frac{2\beta(1 - c_1) M_2}{\rho}\right). \tag{29}$$

The condition (28) does not require the existence of a descent direction. Instead, as long there is a direction whose angle with the gradient is bounded away from being orthogonal, we can ensure that the update direction $\boldsymbol{p}_k$ is a descent one.

Now we discuss the convergence rates of the proposed algorithm. We start with the nonconvex case.

Theorem 3 *Assume* $f(\boldsymbol{w})$ *satisfies Assumption* 1. *For an algorithm that iteratively solves* (25) *that satisfies the conditions* (26) *and* (28) *to obtain* $\boldsymbol{p}_k$ *and uses it as the update direction with backtracking line search to find a step size satisfying* (24), *then the minimum of the norm square of the gradients*

$$\min_{0 \leq J \leq k} \|\nabla f(\boldsymbol{w}_J)\|^2$$

vanishes at a $o(1/k)$ *rate, and* $\|\nabla f(\boldsymbol{w}_k)\|^2$ *converges to zero.*

Next, we consider the convex case.

Theorem 4 *Consider the same conditions in Theorem 3 and assume additionally that f is convex, and given any $\boldsymbol{w}_0$, the value*

$$R_0 := \max_{\boldsymbol{w}: f(\boldsymbol{w}) \le f(\boldsymbol{w}_0)} \min_{\boldsymbol{w}^* \in \Omega} \|\boldsymbol{w} - \boldsymbol{w}^*\| \tag{30}$$

is finite. We then have that $f(\boldsymbol{w}_k)$ converges to f^* at a global rate of $O(1/k)$:

$$f(\boldsymbol{w}_k) - f^* \le \frac{\rho M_1^2 R_0^2}{2M_1^2 + \rho M_2 \delta^2 c_1 \sum_{T=0}^{k-1} \theta_T} \le \frac{\rho M_1^2 R_0^2}{2M_1^2 + \rho M_2 \delta^2 c_1 k \bar{\theta}}. \tag{31}$$

The first inequality in (31) provides the reason why we conduct line search instead of directly applying $\bar{\theta}$ as the step size. Note that our assumption of a finite R_0 in (30) is weaker, in allowing an unbounded Ω, than assuming bounded level sets. On the other hand, the stronger assumption of bounded level set leads to a better convergence rate of $o(1/k)$ as shown in [3, Proposition 1.3.3]. In comparison to Theorem 1, instead of (17) that requires a certain quality of subproblem solutions, Theorem 4 uses the simple condition of a bounded R_0 for a wider applicability, and it results in a $O(1/k)$ convergence rate that is slower than the $O(1/k^2)$ one in Theorem 1.

Finally, we discuss the case in which linear convergence can be obtained.

Theorem 5 *Consider the same assumptions in Theorem 3 and assume additionally that $f(\boldsymbol{w})$ satisfies the Polyak-Łojasiewicz condition [9, 15, 22] for some $\sigma > 0$, i.e.,*

$$\|\nabla f(\boldsymbol{w})\|^2 \ge 2\sigma (f(\boldsymbol{w}) - f^*), \quad \forall \boldsymbol{w}, \tag{32}$$

then the function values converge globally Q-linearly to f^*:

$$\begin{aligned} f(\boldsymbol{w}_{k+1}) - f^* &\le \left(1 - \frac{2\sigma M_2 \delta^2 \theta_k c_1}{M_1^2}\right) \left(f(\boldsymbol{w}_k) - f^*\right) \\ &\le \left(1 - \frac{2\sigma M_2 \delta^2 \bar{\theta} c_1}{M_1^2}\right) \left(f(\boldsymbol{w}_k) - f^*\right), \forall k \ge 0, \end{aligned}$$

where $\bar{\theta}$ is the lower-bound of the step size in (29).

Notice that (32) does not require convexity, and hence even on some nonconvex problems, we can get linear convergence to the global optimum.

To obtain global convergence rates, we used parameters M_1, M_2, R_0 and ρ that are global. However, those values tend to be extreme values that barely occur in practice, thus we often observe much better convergence rates locally. Moreover, when the curvature information of f is properly included in H_k, we tend to observe step sizes far away from the lower bound $\bar{\theta}$ and unit step size is often accepted.

The theorems above provide convergence not only for our algorithm, but also many others, such as the Barzilai-Borwein method [2] with a line search to ensure sufficient objective decrease and line-search Newton methods. Similarly, the algorithm of combining past gradients in [26] can also be treated as a special case of our framework.

5 Implementation details

In this section, we discuss the selection for P_k to improve the empirical performance of our algorithm, and then describe a general problem class whose structure we can utilize to make the implementation even more efficient.

5.1 Choices of the common directions

The convergence analysis in Sect. 4 only suggested that one should include some vectors that are gradient-related. Theorem 1 and Propositions 1-2 suggest that including the current iterate $\boldsymbol{w}_k$ and a momentum vector related to $\boldsymbol{v}_k$ defined in (7) might achieve better theoretical convergence speed. However, when we do not know ρ and σ, we are unable to calculate $\boldsymbol{v}_k$ accurately. In many accelerated first-order methods, instead of this deliberately chosen $\boldsymbol{v}_k$, the update directions from previous iterations are used as "momentum terms" to be combined with the current gradient to form the new update step. Thus previous update directions and the current gradient are the most natural choices for the common directions.

When all previous gradients from the first iteration on are included in P_k, [26] show that we can even achieve asymptotic quadratic convergence. This result suggests that the previous gradients are also good choices to include. When we have a fixed m, how to balance the number of previous gradients and previous update steps is a question. We observe that for quasi-Newton methods belonging to the Broyden class, when the initial estimate for the Hessian is a multiple of the identity, each quasi-Newton step is a linear combination of the current gradient, the difference of the previous gradients in the form $\nabla f(\boldsymbol{w}_{i+1}) - \nabla f(\boldsymbol{w}_i)$, and the previous update steps; see, for example, [21, Chapter 6]. For their limited-memory versions such as L-BFGS [14], the same number of previous updates and previous gradients are used together with the current gradient. As L-BFGS is quite popular in practice, we adopt this choice to use the same number of previous steps and gradient differences. We also take the current iterate as one column of P_k because it is already available.

Assume m is even, now for the kth iteration, we have chosen to use $\boldsymbol{w}_k$, $\nabla f(\boldsymbol{w}_k)$, and the pairs $(\boldsymbol{p}_i, \nabla f(\boldsymbol{w}_{i+1}) - \nabla f(\boldsymbol{w}_i))$ for $i = k-1, k-2, \ldots, k-(m/2-1)$ to form P_k. From (25), we see that when H_k in Algorithm 2 is not damped by a multiple of identity, using $\boldsymbol{p}_i$, $i = k-1, k-2, \ldots, k-(m/2-1)$ together with $\boldsymbol{w}_k$ is equivalent to using $\boldsymbol{w}_i$, $i = k, k-1, \ldots, k-(m/2-1)$, as they will result in the same span and therefore the same $\boldsymbol{p}_k$. Similarly, using the difference of the gradients is equivalent to directly using the gradients. Therefore, our construction of P_k simplifies to

$$P_k = \left[\boldsymbol{w}_{k-(\frac{m}{2}-1)}, \nabla f\left(\boldsymbol{w}_{k-(\frac{m}{2}-1)}\right), \ldots, \boldsymbol{w}_k, \nabla f\left(\boldsymbol{w}_k\right)\right]. \tag{33}$$

This makes the update of P_k straightforward – we just add the current iterate-gradient pair, and then discard the oldest pair when the number of columns in P_k is larger than m. It has been shown in our preliminary report [10] that using previous gradient differences and update steps simultaneously gives better empirical performance than using only one of them.

Another choice is to include in P_k some approximation of the Newton step that can be cheaply obtained to span a subspace whose distance to the solution set might be closer. One such case is taking the diagonal entries of the (generalized) Hessian to form the diagonal matrix $D(\boldsymbol{w}_k)$, and then use the vector $D(\boldsymbol{w}_k)^{-1}\nabla f(\boldsymbol{w}_k)$ as an approximated Newton step. In this case, assuming m is a multiple of three, we take

$$P_k = \left[\boldsymbol{w}_{k-(\frac{m}{3}-1)}, \nabla f\left(\boldsymbol{w}_{k-(\frac{m}{3}-1)}\right), D\left(\boldsymbol{w}_{k-(\frac{m}{3}-1)}\right)^{-1}\nabla f\left(\boldsymbol{w}_{k-(\frac{m}{3}-1)}\right), \ldots, \boldsymbol{w}_k, \nabla f\left(\boldsymbol{w}_k\right), D\left(\boldsymbol{w}_k\right)^{-1}\nabla f\left(\boldsymbol{w}_k\right)\right]. \tag{34}$$

We will compare the empirical performance of (33) and (34) in Sect. 6.

5.2 Problem structure for efficient implementation

The part of forming the linear system (23) is the major additional cost of our algorithm in comparison to first-order algorithms. We will show that when the problem is of the form

$$f\left(\boldsymbol{w}\right) = Q(\boldsymbol{w}) + g\left(X^\top \boldsymbol{w}\right), \tag{35}$$

where Q is a real and quadratic function, $X = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_l) \in \mathbf{R}^{n\times l}$, and g is separable in the sense $g(\boldsymbol{z}) = \sum_{i=1}^{l} g_i(z_i)$, highly efficient implementation is possible for the choices of P_k in (33) or (34). We further write Q in the form

$$Q(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^\top A\boldsymbol{w} + \boldsymbol{b}^\top \boldsymbol{w} \tag{36}$$

for some symmetric matrix $A \in \mathbf{R}^{n\times n}$ and vector $\boldsymbol{b} \in \mathbf{R}^n$. Problems of the form (35) are widely seen in applications including machine learning and signal processing.

The key is to note that for any $k > 0$, P_k and P_{k+1} have overlapping columns and only few columns are updated, and we can thus denote

$$P_{k+1} = [\hat{V}_k, \boldsymbol{w}_{k+1}, V_{k+1}], \tag{37}$$

where $\hat{V}_k$ are the columns also appear in P_k and V_{k+1} are the columns only appear in P_{k+1} excluding $\boldsymbol{w}_{k+1}$. Using this notation, our discussion below can cover any choices of P_k as long as existing columns are reused (and the discussion involving $\boldsymbol{w}_{k+1}$ can be skipped if $\boldsymbol{w}_{k+1}$ is not a column of P_{k+1}).

Throughout the iterations, we will maintain

$$X^\top P_k, \quad X^\top \boldsymbol{w}_k, \quad P_k^\top A P_k, \quad \text{and} \quad P_k^\top \boldsymbol{b}, \tag{38}$$

and discuss their usage and update below. We will assume that the computation of both $X^\top \boldsymbol{v}$ for any $\boldsymbol{v} \in \mathbf{R}^n$ and $X^\top \boldsymbol{u}$ for any $\boldsymbol{u} \in \mathbf{R}^l$ cost $O(T_X)$ for some T_X (the

most common scenarios include that $T_X = ln$ when X is dense and that T_X equals the number of nonzero elements in X when X is sparse), and similarly assume that computing $A\boldsymbol{v}$ for $\boldsymbol{v} \in \mathbf{R}^n$ costs $O(T_A)$. Under Assumption 1, (35) is twice-differentiable almost everywhere. Its gradient and (generalized) Hessian [16] are respectively

$$\nabla f(\boldsymbol{w}) = A\boldsymbol{w} + \boldsymbol{b} + X\boldsymbol{u}_{\boldsymbol{w}}, \quad \nabla^2 f(\boldsymbol{w}) = A + XD_{\boldsymbol{w}}X^\top, \tag{39}$$

where $D_{\boldsymbol{w}}$ is a diagonal matrix, and

$$(D_{\boldsymbol{w}})_{i,i} := \partial^2 g_i(\boldsymbol{x}_i^\top \boldsymbol{w}), \quad (u_{\boldsymbol{w}})_i := \partial g_i(\boldsymbol{x}_i^\top \boldsymbol{w}), \quad i = 1, \ldots, l. \tag{40}$$

The main computation at each iteration of Algorithms 1 and 2 is to construct and solve the linear system (23). For problems of the form (35), if at the kth iteration, the iterate is $\boldsymbol{w}_k$ and the linear function (4) is defined by P_k, then the matrix on the left-hand side of (23) is

$$P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k = P_k^\top A P_k + P_k^\top X D_{\boldsymbol{w}_k} X^\top P_k. \tag{41}$$

For large-scale problems, one should maintain $X^\top P_k$ and calculate the second term in (41) by $(X^\top P_k)^\top D_{\boldsymbol{w}_k}(X^\top P_k)$ instead of $P_k^\top (X D_{\boldsymbol{w}_k} X^\top) P_k$, where the latter requires $O(n^2)$ storage and $O(n^2 l)$ computation to explicitly form the Hessian matrix of $g(X^\top \boldsymbol{w})$. Further, $X^\top P_k$ does not need to be calculated from scratch (which has an expensive $O(lmn)$ cost) because we can take the property that P_k and P_{k-1} share most columns to efficiently update $X^\top P_{k-1}$ to $X^\top P_k$. We should also maintain $X^\top \boldsymbol{w}_k$ (which is actually a column of $X^\top P_k$ for our choice of P_k) so that the cost of computing $D_{\boldsymbol{w}_k}$ can be reduced from the original $O(T_X)$ to $O(l)$. The second term of (41) can thus be efficiently computed in $O(m^2 l)$ time, which is often even cheaper than computing the gradient of $g(X^\top \boldsymbol{w}_k)$ that costs $O(T_X)$.

For maintaining $X^\top P_k$, we have from (37) that

$$X^\top P_k = \left[X^\top \hat{V}_{k-1}, X^\top \boldsymbol{w}_k, X^\top V_k \right]. \tag{42}$$

The term $X^\top \hat{V}_{k-1}$ is directly available because it is a submatrix of $X^\top P_{k-1}$ maintained in the $(k-1)$th iteration, and $X^\top \boldsymbol{w}_k$ is obtained from the previous iteration in the line search procedure that we will explain later when discussing (48). Finally, from (33) and (34), V_k in (37) has only one or two columns for all k, so computing $X^\top V_k$ and therefore maintaining XP_k costs only $O(T_X)$.

Next, we consider the first term in (41) and use (37) to deduce that

$$P_k^\top A P_k = P_k^\top \left[A\hat{V}_{k-1}, \ A\boldsymbol{w}_k, \ AV_k \right]. \tag{43}$$

We compute AV_k in $O(T_A)$ cost, and then $P_k^\top (AV_k)$ costs only $O(mn)$. To compute $P_k^\top A\boldsymbol{w}_k$, in addition to (38), from the previous iteration we keep track of θ_{k-1}, $\boldsymbol{t}_{k-1}$,

(44), and the update direction

$$\boldsymbol{p}_{k-1} = P_{k-1}\boldsymbol{t}_{k-1}. \tag{44}$$

Then the following calculation is conducted.

$$\begin{aligned} P_k^\top A\boldsymbol{w}_k &= P_k^\top A\left(\boldsymbol{w}_{k-1} + \theta_{k-1}P_{k-1}\boldsymbol{t}_{k-1}\right) = P_k^\top A\boldsymbol{w}_{k-1} + \theta_{k-1}P_k^\top AP_{k-1}\boldsymbol{t}_{k-1} \\ &= \begin{bmatrix} \hat{V}_{k-1}^\top A\boldsymbol{w}_{k-1} \\ \boldsymbol{w}_k^\top A\boldsymbol{w}_{k-1} \\ \left(V_k^\top A\right)\boldsymbol{w}_{k-1} \end{bmatrix} + \theta_{k-1}\begin{bmatrix} \left(\hat{V}_{k-1}^\top AP_{k-1}\right)\boldsymbol{t}_{k-1} \\ \left(\boldsymbol{w}_k^\top AP_{k-1}\right)\boldsymbol{t}_{k-1} \\ \left(V_k^\top A\right)\left(P_{k-1}\boldsymbol{t}_{k-1}\right) \end{bmatrix}, \end{aligned} \tag{45}$$

where, if possible, we use parentheses to specify the details of operations that will be explained below. For the first row in (45), both $\hat{V}_{k-1}^\top A\boldsymbol{w}_{k-1}$ and $\hat{V}_{k-1}^\top AP_{k-1}$ are entries in $P_{k-1}^\top AP_{k-1}$ maintained, and the inner product between the latter and $\boldsymbol{t}_{k-1}$ costs only $O(m)$. For the third row in (45) that involves $V_k^\top AP_{k-1}\boldsymbol{t}_{k-1}$ and $V_k^\top A\boldsymbol{w}_{k-1}$, we use AV_k computed above and $P_{k-1}\boldsymbol{t}_{k-1}$ available in (44), and the remaining inner products cost only $O(n)$. For the second row in (45), we note that

$$\begin{aligned} \boldsymbol{w}_k^\top A\boldsymbol{w}_{k-1} &= \left(\boldsymbol{w}_{k-1} + \theta_{k-1}P_{k-1}\boldsymbol{t}_{k-1}\right)^\top A\boldsymbol{w}_{k-1} \\ &= \boldsymbol{w}_{k-1}^\top A\boldsymbol{w}_{k-1} + \theta_{k-1}\boldsymbol{t}_{k-1}^\top\left(P_{k-1}^\top A\boldsymbol{w}_{k-1}\right) \end{aligned}$$

involves two entries in $P_{k-1}^\top AP_{k-1}$ and the computation costs only $O(m)$ for the inner product between $\boldsymbol{t}_{k-1}$ and $P_{k-1}^\top A\boldsymbol{w}_{k-1}$, which is exactly $\boldsymbol{w}_k^\top AP_{k-1}\boldsymbol{t}_{k-1}$ needed in (45). Finally, $P_k^\top A\hat{V}_{k-1}$ in (43) can be decomposed into

$$P_k^\top A\hat{V}_{k-1} = \begin{bmatrix} \hat{V}_{k-1}^\top A\hat{V}_{k-1} \\ \boldsymbol{w}_k^\top A\hat{V}_{k-1} \\ V_k^\top A\hat{V}_{k-1} \end{bmatrix}. \tag{46}$$

The first entry is available from $P_{k-1}^\top AP_{k-1}$, and the rest two have been calculated above ($\boldsymbol{w}_k^\top A\hat{V}_{k-1}$ from the first row in (45) and $V_k^\top A\hat{V}_{k-1}$ as entries of $P_k^\top(AV_k)$ in (43)), so these entries are obtained with no additional cost. Therefore, the cost of maintaining the term $P_k^\top AP_k$ using information from $P_{k-1}^\top AP_{k-1}$ is $O(T_A + mn)$.

Next is the right-hand side of (23), which from (39) can be calculated by

$$P_k^\top\nabla f(\boldsymbol{w}_k) = P_k^\top A\boldsymbol{w}_k + P_k^\top\boldsymbol{b} + \left(X^\top P_k\right)^\top\boldsymbol{u}_{\boldsymbol{w}_k}. \tag{47}$$

The cost is low because $X^\top P_k$ and $P_k^\top A\boldsymbol{w}_k$ have been respectively calculated in (42) and (45), and we only need to compute $\boldsymbol{u}_{\boldsymbol{w}_k}$, $\left(X^\top P_k\right)^\top\boldsymbol{u}_{\boldsymbol{w}_k}$, and $P_k^\top\boldsymbol{b}$. As indicated in (38), $P_k^\top\boldsymbol{b}$ should be maintained because we can reuse some elements of $P_{k-1}^\top\boldsymbol{b}$ and only need to calculate $V_k^\top\boldsymbol{b}$.[3] The cost of maintaining $P_k^\top\boldsymbol{b}$ is thus only $O(n)$. The

[3] The update of $\boldsymbol{w}_k^\top\boldsymbol{b}$ is similar to (48) and only costs $O(m)$.

calculation of $\boldsymbol{u}_{\boldsymbol{w}_k}$ is $O(l)$ because $X^\top \boldsymbol{w}_k$ is obtained from the previous iteration in (48), and the part of $\left(X^\top P_k\right)^\top \boldsymbol{u}_{\boldsymbol{w}_k}$ costs only $O(ml)$. This means that the right-hand side of (23) is not the bottleneck.

When we conduct line search in (24), the above calculated information can be utilized to reduce the cost. The term $\nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k$ can be calculated between (47) and $\boldsymbol{t}_k$ with $O(m)$ cost. All other terms are already known (the previous objective value is maintained) so we just need to discuss how to evaluate $f(\boldsymbol{w}_k + \beta^i \boldsymbol{p}_k)$ with multiple values of i efficiently. For $g(X^\top \boldsymbol{w})$, we can use $X^\top P_k$ obtained in (42) to calculate $(X^\top P_k)\boldsymbol{t}_k$ in $O(lm)$ time. Thus

$$X^\top \boldsymbol{w}_k + \beta^i \left(X^\top P_k\right) \boldsymbol{t}_k \tag{48}$$

for each i can be obtained in $O(l)$ time. Therefore, for each i, $g(X^\top(\boldsymbol{w} + \beta^i \boldsymbol{p}))$ can be evaluated in $O(l)$ time as well. For the quadratic part, we see that

$$\begin{aligned} &\left(\boldsymbol{w}_k + \beta^i P_k \boldsymbol{t}_k\right)^\top A \left(\boldsymbol{w}_k + \beta^i P_k \boldsymbol{t}_k\right) \\ &\quad = \boldsymbol{w}_k^\top A \boldsymbol{w}_k + 2\beta^i \boldsymbol{w}_k^\top A P_k \boldsymbol{t}_k + \beta^{2i} \boldsymbol{t}_k^\top \left(P_k^\top A P_k\right) \boldsymbol{t}_k. \end{aligned} \tag{49}$$

As $P_k^\top A P_k$ is maintained, $\boldsymbol{w}_k^\top A \boldsymbol{w}_k$ is one of its elements, and $\boldsymbol{w}_k^\top A P_k \boldsymbol{t}_{k-1}$ has been calculated in (45), we just need $O(m^2)$ overhead for the last term. Then each line search step costs only $O(1)$ on this part. Note that the sum in (48) with the final β^i accepted will be $X^\top \boldsymbol{w}_{k+1}$ needed for the next iteration; see (38). The new function value $f(\boldsymbol{w}_{k+1})$ is also obtained.

Interestingly, the line search procedure via (48) and (49) does not generate the next iterate $\boldsymbol{w}_{k+1}$. Thus, in the end of the iteration we calculate the update direction

$$\boldsymbol{p}_k = P_k \boldsymbol{t}_k, \tag{50}$$

which as shown in (44) and (45) will be used in the next iteration, and finally obtain

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \theta_k \boldsymbol{p}_k. \tag{51}$$

We summarize in Algorithm 3 how we maintain additional information to make the implementation for (35) more efficient and show the corresponding cost of each computational step.

The problem structure (35) also allows us to conduct multiple inner iterations with a cost much lower than parts other than solving the subproblem in an outer iteration, including updating P_k and calculating the full gradient. We explain that as long as P_k remain unchanged, the coefficients in the Newton linear systems can be evaluated quickly. From (6), (40), and (41), if $\boldsymbol{w}$ is changed in an inner iteration, all we need to calculate for updating the coefficients in the Newton linear system are $D_{\boldsymbol{w}}$ and $(X^\top P_k)^\top D_{\boldsymbol{w}} (X^\top P_k)$ for the matrix, and $\boldsymbol{u}_{\boldsymbol{w}}$, $(P_k^\top A)\boldsymbol{w}$, $(X^\top P_k)^\top \boldsymbol{u}_{\boldsymbol{w}}$ for $P_k^\top \nabla f(\boldsymbol{w})$. For the data-related part $g(\boldsymbol{z})$ with $\boldsymbol{z} = X^\top \boldsymbol{w}$, the objective value, $\boldsymbol{u}_{\boldsymbol{w}}$, and $D_{\boldsymbol{w}}$ can

Algorithm 3: Efficient implementation of Algorithm 2 for (35).

Given $\boldsymbol{w}_0 \in \mathbf{R}^n$ and matrices A and X such that computing $A\boldsymbol{v}$ and $X^\top \boldsymbol{v}$ respectively cost $O(T_A)$ and $O(T_X)$
Compute $X^\top \boldsymbol{w}_0$, $A\boldsymbol{w}_0$, and $f(\boldsymbol{w}_0)$
for k=0,1,... **do**
  Use $X^\top \boldsymbol{w}_k$ to compute $\nabla f(\boldsymbol{w}_k)$ by (39) ▷ $O(T_X + T_A + n + l)$
  Update P_k in the form (37)
  Compute $X^\top P_k$ (using $X^\top P_{k-1}$, $\boldsymbol{t}_{k-1}$ and θ_{k-1} if $k > 0$)
    – $X^\top \hat{V}_{k-1}$ ▷ $O(1)$
    – $X^\top \boldsymbol{w}_k$ is available from (48) ▷ $O(1)$
    – $X^\top V_k$ ▷ $O(T_X)$*
  Compute $P_k^\top A P_k$ (through (43) using $P_{k-1}^\top A P_{k-1}$, $\boldsymbol{t}_{k-1}$ and θ_{k-1} if $k > 0$)
    – AV_k and then $P_k^\top (AV_k)$ ▷ $O(T_A + mn)$*
    – $P_k^\top A\boldsymbol{w}_k$ by (45) ▷ $O(m + n)$
    – $P_k^\top A\hat{V}_{k-1}$ by (46) ▷ $O(1)$
  Compute $P_k^\top \boldsymbol{b}$ (using $\boldsymbol{t}_{k-1}$, θ_{k-1} and $P_{k-1}^\top \boldsymbol{b}$ if $k > 0$) ▷ $O(n)$
  Use $X^\top \boldsymbol{w}_k$ to calculate (40)
  Construct the linear system (23) by (41) and (47) ▷ $O(m^2 l + n)$
  Solve the system (23) for $\boldsymbol{t}_k$ ▷ $O(m^3)$
  Compute $\boldsymbol{t}_k^\top (P_k^\top A P_k)\boldsymbol{t}_k$ and $(X^\top P_k)\boldsymbol{t}_k$ respectively for (49) and (48) ▷ $O(l + m^2)$
  Conduct backtracking line search to find θ_k that satisfies (24) using (49) and (48) and record $f(\boldsymbol{w}_{k+1})$
  Compute $\boldsymbol{p}_k$ by (50) and $\boldsymbol{w}_{k+1}$ by (51) ▷ $O(mn)$
end
*Note that V_k constructed by (33) or (34) has at most 2 columns, so the cost of $X^\top V_k$ and AV_k are indeed $O(T_X)$ and $O(T_A)$.

all be quickly calculated in $O(l)$ time through $\boldsymbol{z}$ that is maintained by (48). Notice that in this case, the column of $\boldsymbol{w}_k$ in (37) is the iterate from the latest outer iteration, but not the latest iterate from the inner iteration (so are other columns in P_k such as $\nabla f(\boldsymbol{w}_k)$), so the update of $\boldsymbol{w}$ and $\boldsymbol{z}$ are disentangled from the update of P_k, where the former two change every inner iteration but the last one is updated every outer iteration. Thus, when P_k and therefore $X^\top P_k$ remains unchanged, we can compute $(X^\top P_k)^\top D_{\boldsymbol{w}} (X^\top P_k)$ in $O(lm^2)$ time. For (47), from (45), $P_k^\top A\boldsymbol{w}$ can be updated with $O(m+n)$ cost, and with both $\boldsymbol{u}_{\boldsymbol{w}}$ and $X^\top P_k$ available, $(X^\top P_k)^\top \boldsymbol{u}_{\boldsymbol{w}}$ costs only $O(lm)$ to compute. Thus constructing and solving the linear system can be much cheaper than, for example, calculating $X^\top V_k$ when P_k is updated in an outer iteration. We have therefore explained why (35) allows us to solve (2) with multiple inner Newton iterations efficiently. For updating (38) after one outer iteration, we accumulate the values of $\boldsymbol{t}$ in all inner iterations within the same outer iteration to conduct the updates in the manner discussed above.

5.2.1 Summary of cost analysis

We summarize the cost of the major steps in Algorithm 3 in Table 1. In total, the computational cost per iteration is

Table 1 Summarization of the cost of the major computational steps of Algorithm 3

Step	Cost
Compute $\nabla f(\boldsymbol{w}_k)$ using $A\boldsymbol{w}_k$ and $X^\top \boldsymbol{w}_k$	$O(T_X + T_A + n + l)$
$X^\top P_k$	$O(T_X)$
$P_k^\top A P_k$	$O(T_A + mn)$
$P_k^\top \boldsymbol{b}$	$O(n)$
(40)	$O(l)$
Compute (41) and (47)	$O(m^2 l + n)$
Solve (23)	$O(m^3)$
Line search	$O(l + m^2 + l \times \#(\text{line search steps}))$
Compute $\boldsymbol{p}_k$ and $\boldsymbol{w}_{k+1}$	$O(mn)$

$$O\left(T_X + T_A + m^2 l + l \times \#\,(\text{line search steps}) + mn + m^3\right), \tag{52}$$

and the number of line search steps is upper-bounded by a constant according to Theorem 2 while in practice we observe that unit step size is often accepted. In general, m tends to a small constant such that $O(m^3) = O(1)$ and the dominant term is often $T_X + T_A + m^2 l$. For example, if A and X are both dense, $T_X + T_A = O(ln)$. When m and the number of backtracking steps are not large, the last three terms in (52) are clearly dominated by $T_X + T_A$. If X or A are sparse, unless the matrix is highly sparse such that each row or column has a small number of non-zeros, the dominant term is still $T_X + T_A + m^2 l$.

In comparison with the general cost analysis given in Sect. 3.1, from (39) and (40), we note that $T_X + T_A$ is exactly $\mathrm{T}_{\mathrm{grad}}$ (assuming $l + n = O(T_X + T_A)$ as argued above). Therefore, while the plain implementation involves $m(T_X + T_A)$, Algorithm 3 by utilizing the problem structure requires only two to three $(T_x + T_A)$, as can be seen from (52). As $T_X + T_A$ is usually dominant, the efficient implementation described above can indeed lead to significant improvement in the computational cost.

6 Numerical experiments

We present numerical results of Algorithms 1 and 2 running with a single core of a machine with 16GB memory. Code for reproducing our results in this section and Sects. 7–8 is available at https://www.csie.ntu.edu.tw/~cjlin/papers/l-commdir/l-commdir-journal-exp.tar.gz.

Throughout the experiments, we consider ℓ_2-regularized logistic regression and ℓ_2-regularized squared-hinge loss SVM problems, which are of the forms

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l} \log\left(1 + \exp\left(y_i \boldsymbol{w}^\top \boldsymbol{x}_i\right)\right), \quad \text{and} \tag{53}$$

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l} \max\left\{1 - y_i \boldsymbol{w}^\top \boldsymbol{x}_i, 0\right\}^2, \tag{54}$$

respectively, where $C > 0$ is a pre-specified parameter, $\boldsymbol{x}_i, i = 1, \ldots, l$ are the feature vectors of the training data with corresponding labels $y_i \in \{-1, 1\}$. Notice that both problems are of the form (35), with $A = I$ and $\boldsymbol{b} = \boldsymbol{0}$ in the quadratic part, and $g_i(z_i)$ is the summand times C.

Both problems satisfy Assumption 1 and the ℓ_2 regularization makes the problem strongly convex. Thus Theorem 5 and the first case of Theorem 1 apply because σ-strong convexity implies (32) with the same σ. The datasets we use are summarized in Table 2.[4] We compare the relative difference to the optimal objective value:

$$\left|\frac{f(\boldsymbol{w}) - f^*}{f^*}\right|, \tag{55}$$

where f^* is obtained by running our algorithm long enough. All methods are implemented in C/C++.

We first examine the optimal convergence speed described in Theorem 1 and compare Algorithm 1 with related approaches. To apply Theorem 1, we consider Proposition 1 and take

$$P_k = [\nabla f(\boldsymbol{w}_k), \boldsymbol{v}_k, \boldsymbol{w}_k] \tag{56}$$

at each iteration to include the span of $\nabla f(\boldsymbol{w}_k)$ and $\tilde{\boldsymbol{v}}_{k+1}$, as required by Proposition 1. We use this choice instead of directly including $\tilde{\boldsymbol{v}}_{k+1}$ because $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are maintained throughout for calculating $\boldsymbol{v}_{k+1}$, so we can utilize these two vectors easily. The subproblem (14) is solved by the Newton method described in Sect. 3, with the inner stopping condition being (21). Notice that classical analysis for line-search Newton guarantees that the inner iterates will approach the solution set of (2), and Proposition 1 ensures that when the iterate is close enough to the solution set, this condition will be satisfied. Therefore, (21) is a valid stopping condition.

We use the (smaller) first four datasets listed in Table 2 in this experiment. For parameters estimation, it is clear that for (53) and (54) the quadratic term is 1-strongly convex so we always have $\sigma = 1$. For ρ, clearly the gradient of the quadratic term is 1-Lipschitz continuous, and in the data-related term, each $g_i(\boldsymbol{x}_i^\top \boldsymbol{w})$ has a $C\hat{\rho}\|\boldsymbol{x}_i\|_2^2$-Lipschitz continuous gradient with $\hat{\rho} = 0.25$ for (53) and $\hat{\rho} = 2$ for (54). We thus set ρ as the following upper bound.

$$\rho = 1 + C\sum_{i=1}^{l} \hat{\rho}\|\boldsymbol{x}_i\|_2^2. \tag{57}$$

We compare the following methods.

[4] All except yahoo-japan and yahoo-korea are downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

Table 2 Data statistics

Dataset	#instances	#features	Density (%)
a9a	32,561	123	11.2757
real-sim	72,309	20,958	0.2448
news20	19,996	1,355,191	0.0336
rcv1t	677,399	47,236	0.1549
yahoo-japan	140,963	832,026	0.0160
yahoo-korea	368,444	3,053,939	0.0111

- L-CommDir-Optimal: Algorithm 1 with the settings described above.
- Accelerated gradient (AG) [18, 20]: we use the same $\sigma = 1$ and the ρ estimation in (57).
- CommDir [26]: the unlimited-memory version of Algorithm 3 that takes all gradients up to the current iteration to form P in (2).
- L-CommDir-BFGS: Algorithm 3 with (33).
- L-CommDir-Diag: Algorithm 3 with (34).

We use the suffix "-t" for L-CommDir-BFGS and L-CommDir-Diag to indicate that information from the latest t iterations (including the current iteration) is used. We mainly use $t = 5$, making $m = 10$ for (33) in L-CommDir-BFGS and $m = 15$ in (34) for L-CommDir-Diag. To compare with L-CommDir-Optimal, we also consider $t = 1$ (information from the current iteration only), making $m = 2$ and $m = 3$ respectively for L-CommDir-BFGS and L-CommDir-Diag. For all methods that require line search to satisfy (24), we use the fixed parameters $\beta \equiv 0.5$ and $c_1 \equiv 10^{-2}$ throughout in both this section and all following experiments.

The results are shown in Table 3. We see that L-CommDir-Optimal always needs only one Newton step to satisfy (22), making Algorithm 1 with the inner stopping condition being (15) equivalent to Algorithm 2 under P_k in (56) in this experiment. Except for a9a, L-CommDir-Optimal always outperforms AG significantly, while when AG is faster, L-CommDir-Optimal takes at most four times of the number of iterations, confirming that our method is converging at least as fast. We also see that when $t = 1$, L-CommDir-BFGS and L-CommDir-Diag tend to be worse than L-CommDir-Optimal, indicating that a momentum term is indeed helpful. However, when $t \geq 5$, L-CommDir-BFGS, L-CommDir-Diag, and CommDir all converge much faster than these optimal first-order methods empirically in terms of both the running time and the number of iterations. Note that L-CommDir-BFGS always takes no fewer iterations than CommDir, as the subspace considered by the former is always in that of the latter, but L-CommDir-Diag sometimes converges faster. Notice that L-CommDir-BFGS and CommDir can still be considered as first-order methods in the sense that $\boldsymbol{p}_k \in \mathtt{span}\,(\boldsymbol{w}_0, \nabla f(\boldsymbol{w}_0), \ldots, \nabla f(\boldsymbol{w}_k))$, but they show much faster empirical performance.[5] Hence, we focus on using (33) and (34) for P in the sequel.

[5] Obviously, $\boldsymbol{p}_k \in \mathtt{span}\,(\boldsymbol{w}_0, \nabla f(\boldsymbol{w}_0), \boldsymbol{w}_1, \nabla f(\boldsymbol{w}_1), \ldots, \boldsymbol{w}_k, \nabla f(\boldsymbol{w}_k))$, and by simple induction, we can conclude that each $\boldsymbol{w}_k \in \mathtt{span}\left(\boldsymbol{w}_0, \nabla f(\boldsymbol{w}_0), \nabla f(\boldsymbol{w}_1), \ldots, \nabla f(\boldsymbol{w}_{k-1})\right)$ for all k. Therefore, $\boldsymbol{p}_k$ is in the span of the previous gradients and the initial point.

Table 3 Single-core comparison of different methods on (53) and (54) with $C = 1$

Problem		(53)				(54)			
Data		a9a	real-sim	news20	rcv1t	a9a	real-sim	news20	rcv1t
AG	Time	30.5	50.6	118.8	3275.1	36.1	68.0	315.1	4801.3
	Iter.	2355	1295	635	4145	5780	4248	2241	12032
Optimal	Time	60.5	3.5	11.5	241.6	169.3	11.9	71.9	1428.0
	(Outer) Iter.	3488	51	46	265	20726	385	357	3127
	Largest inner iter.	1	1	1	1	1	1	1	1
BFGS-1	Time	85.4	5.3	14.0	468.8	304.5	22.3	194.6	2612.0
	Iter.	5995	88	72	564	41008	748	1285	6094
DIAG-1	Time	292.9	6.3	40.1	453.1	796.3	23.7	288.0	1323.0
	Iter.	13577	60	104	296	58780	415	958	1589
BFGS-5	Time	2.1	1.4	3.5	44.3	2.4	2.1	9.6	74.5
	Iter.	107	20	13	44	215	59	40	154
DIAG-5	Time	3.6	2.2	6.7	66.2	6.6	2.9	13.9	91.3
	Iter.	109	17	12	35	309	43	28	98
CommDir	Time	3.3	1.4	4.8	45.8	4.9	3.1	19.3	264.3
	Iter.	61	18	13	37	76	40	33	101

We present the required time (in seconds) and number of iterations to make (55) no larger than 10^{-8}. The names Optimal, BFGS, and DIAG omit the prefix L-CommDir. The row "largest inner iter." of L-CommDir-Optimal shows the largest number of inner iterations taken over the outer iterations to solve (2) to satisfy (21)

We continue to compare CommDir and L-CommDir further using all datasets in Table 2. We further try $C \in \{10^{-3}, 1, 10^{3}\}$ to test how the algorithms perform when the condition number changes. We compare $CommDir$ with both L-CommDir-BFGS and L-CommDir-Diag using information from the latest t iterations with $t \in \{5, 10, 15, 20\}$. The results are shown in Tables 4 and 5.

In all situations, CommDir requires fewer iterations than L-CommDir-BFGS. But even with only $t = 5$, the convergence speed of L-CommDir-BFGS is competitive. When it comes to the real running time, however, on difficult problems that take more iterations, especially when $C = 10^{3}$, CommDir can be much slower in later iterations when m becomes large, while L-CommDir does not suffer from this issue and is therefore much faster.

We also observe that L-CommDir-Diag tends to be faster than L-CommDir-BFGS in terms of the iteration count, but the running time might not be as advantageous because computing the diagonal entries of the Hessian is as costly as computing the gradient. Also, from (34), V_{k+1} in (37) has two columns in L-CommDir-Diag, so its updating $X^{\top} P_k$ is more costly than that of L-CommDir-BFGS.

7 Multicore parallelization

In comparison to other first-order and even second-order methods, one advantage of our method is that it can be better parallelized. When the problem is of the form (35) and X is sparse, the bottleneck of first-order methods is the computation of $X^{\top}\boldsymbol{w}$ in (40) for calculating the gradient. This calculation is mainly reading through all the entries of X, so the bottleneck is usually the memory bandwidth instead of the computational power. Therefore, its parallelism is limited. Similar situation applies to second-order methods that repeatedly calculate the Hessian-vector products, as that is also a memory-bound procedure. Therefore, although theoretically all non-stochastic first-order and second-order methods are inherently parallel as the major operations are matrix-vector products, usually we do not experience that much speedup by using more cores.

On the other hand, the additional calculation of (41) in our method is a matrix-matrix calculation and the most expensive part $(X^{\top} P_k)^{\top} D_{\boldsymbol{w}_k} (X^{\top} P_k)$ involves dense matrix-matrix operations, so there are more data reusing and the memory bandwidth is not the bottleneck anymore. Therefore, better parallelism of our method can be expected in multicore environments.

Moreover, in the multicore setting, except the step of solving the linear system, each of other steps in Table 1 invovles a set of independent operations such as matrix-vector products. Thus, they can be easily parallelized. Assume m^3 is relatively small and we do no parallelize it, then if we have K cores, the computational cost of one iteration of Algorithm 3 reduces from (52) to

$$O\left(\frac{T_X + T_A + m^2 l + l \times \#\,(\text{line search steps}) + mn}{K} + m^3\right). \tag{58}$$

Table 4 Single-core comparison of L-CommDir and CommDir on (53)

Data	C		BFGS-5	BFGS-10	BFGS-15	BFGS-20	CommDir	DIAG-5	DIAG-10	DIAG-15	DIAG-20
a9a	0.001	Time	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.2
		Iter.	8	8	8	8	8	7	7	7	7
	1	Time	2.1	3.0	4.4	5.8	3.3	3.6	3.8	4.0	3.9
		Iter.	107	88	82	75	61	109	61	42	34
	1000	Time	22.0	23.8	26.8	37.7	10.4	11.6	7.7	6.6	6.8
		Iter.	1086	666	449	405	98	344	119	63	48
real-sim	0.001	Time	0.2	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
		Iter.	3	3	3	3	3	3	3	3	3
	1	Time	1.4	1.6	1.6	1.4	1.4	2.2	2.4	2.3	2.3
		Iter.	20	18	18	18	18	17	15	15	15
	1000	Time	19.3	24.6	30.9	38.1	51.4	29.3	34.2	47.4	63.7
		Iter.	261	222	188	164	128	216	160	144	134
news20	0.001	Time	0.7	0.7	0.7	0.7	0.8	0.9	0.9	0.9	0.9
		Iter.	3	3	3	3	3	2	2	2	2
	1	Time	3.5	4.1	4.0	4.0	4.8	6.7	7.1	7.2	7.2
		Iter.	13	13	13	13	13	12	11	11	11
	1000	Time	30.9	39.4	43.1	42.1	50.1	85.2	83.8	74.7	71.0
		Iter.	107	97	84	70	55	143	98	69	56
rcv1t	0.001	Time	5.3	5.2	5.3	5.2	5.3	10.1	9.9	9.9	9.9
		Iter.	6	6	6	6	6	6	6	6	6
	1	Time	44.3	53.5	65.9	79.6	45.8	66.2	81.5	97.8	106.5

Table 4 continued

Data	C		BFGS-5	BFGS-10	BFGS-15	BFGS-20	CommDir	DIAG-5	DIAG-10	DIAG-15	DIAG-20
		Iter.	44	39	37	37	37	35	31	29	28
	1000	Time	938.8	1118.0	1513.0	2024.0	9967.9	736.6	825.0	1141.0	1598.0
		Iter.	916	736	662	612	380	382	278	250	239
yahoojp	0.001	Time	1.4	1.4	1.4	1.4	1.5	2.7	2.7	2.7	2.7
		Iter.	4	4	4	4	4	4	4	4	4
	1	Time	18.3	18.1	19.7	21.9	21.8	24.1	24.6	25.8	26.8
		Iter.	46	36	33	33	33	31	25	23	23
	1000	Time	311.5	276.2	288.7	332.5	712.8	369.9	334.2	393.0	450.5
		Iter.	776	506	398	353	204	459	300	264	232
yahookr	0.001	Time	12.9	12.8	12.8	12.8	13.4	21.4	21.2	21.2	21.2
		Iter.	6	6	6	6	6	5	5	5	5
	1	Time	212.6	193.1	196.6	205.1	219.7	256.7	246.6	253.6	260.1
		Iter.	94	72	63	58	55	57	46	41	38
	1000	Time	3805.0	3071.0	3002.0	3244.0	9863.5	3360.0	3411.0	3459.0	3809.0
		Iter.	1674	1113	895	798	377	742	612	504	453

We present the required time and number of iterations to make (55) no larger than 10^{-8}

Table 5 Single-core comparison of L-CommDir and CommDir on (54)

Data	C		BFGS-5	BFGS-10	BFGS-15	BFGS-20	CommDir	DIAG-5	DIAG-10	DIAG-15	DIAG-20
a9a	0.001	Time	0.2	0.3	0.3	0.2	0.2	0.4	0.4	0.4	0.4
		Iter.	19	17	17	17	17	18	15	15	15
	1	Time	2.4	3.6	5.3	7.1	4.9	6.6	6.3	5.0	3.2
		Iter.	215	174	156	139	76	309	157	76	41
	1000	Time	14.9	13.9	18.4	24.2	11.3	4.1	3.0	3.1	3.6
		Iter.	1330	661	519	440	104	193	76	51	44
real-sim	0.001	Time	0.3	0.3	0.3	0.3	0.2	0.5	0.5	0.5	0.5
		Iter.	5	5	5	5	5	5	5	5	5
	1	Time	2.1	2.0	2.2	2.4	3.1	2.9	2.7	2.9	3.1
		Iter.	59	46	42	40	40	43	32	30	29
	1000	Time	35.3	30.9	34.4	38.2	2040.0	170.0	189.9	210.8	250.6
		Iter.	1184	862	798	738	480	2911	2698	2449	2387
news20	0.001	Time	0.9	0.9	0.9	0.9	1.0	1.9	1.9	1.9	1.9
		Iter.	4	4	4	4	4	4	4	4	4
	1	Time	9.6	11.7	13.3	14.8	19.3	13.9	18.0	19.5	21.0
		Iter.	40	35	33	33	33	28	26	24	24
	1000	Time	154.5	221.1	278.1	346.3	2366.6	1025.0	1144.0	1507.0	1850.0
		Iter.	673	630	589	583	400	2132	1528	1475	1427
rcv1t	0.001	Time	9.3	9.8	9.1	9.1	8.4	18.2	17.8	17.3	17.3
		Iter.	12	12	12	12	12	12	11	11	11

Table 5 continued

Data	C		BFGS-5	BFGS-10	BFGS-15	BFGS-20	CommDir	DIAG-5	DIAG-10	DIAG-15	DIAG-20
	1	Time	74.5	74.2	81.4	91.2	264.3	91.3	92.3	98.9	111.4
		Iter.	154	129	119	113	101	98	84	76	73
	1000	Time	1463.0	1490.0	1628.0	1801.0	-	1940.0	2046.0	2402.0	2792.0
		Iter.	3282	2863	2644	2471	-	2213	2006	1992	1949
yahoojp	0.001	Time	3.1	3.1	3.1	3.1	3.4	4.8	4.9	4.9	4.9
		Iter.	9	9	9	9	9	7	7	7	7
	1	Time	49.8	51.8	58.1	65.5	111.6	54.3	56.3	62.0	68.8
		Iter.	182	139	121	111	89	98	74	64	59
	1000	Time	1299.0	1211.0	1321.0	1413.0	–	5882.0	6304.0	6944.0	8178.0
		Iter.	5247	3565	2996	2566	-	11492	8789	7316	6833
yahookr	0.001	Time	25.8	27.4	26.6	26.7	27.9	47.7	45.2	45.2	45.2
		Iter.	12	12	12	12	12	11	10	10	10
	1	Time	569.6	525.6	523.4	550.3	926.4	575.5	585.3	621.9	676.6
		Iter.	304	232	194	174	130	152	127	113	105
	1000	Time	11440.0	10860.0	10730.0	10860.0	–	–	–	–	–
		Iter.	7554	5830	4791	4123	–	–	–	–	–

We present the required time and number of iterations to make (55) no larger than 10^{-8}
"–" indicates that the algorithm fails to reach the desired precision within 4 hours

Table 6 Data statistics

Dataset	#instances	#features	Density (%)
url	2,396,130	3,231,961	0.0036
epsilon	400,000	2000	100.0000
webspam	350,000	16,609,143	0.0224
KDD2010-a	8,407,752	20,216,830	0.0002

We also note that in a shared-memory multicore environment, communication between cores is usually extremely fast and not an issue for the overall running time.

7.1 Multicore experiments

To verify our discussion, we conduct numerical experiments to show the empirical speedup of our method using different number of cores, and compare it with Multicore-LIBLINEAR 2.30 [12],[6] which is a state-of-the-art multicore package for (53). In particular, we compare our method with the trust-region Newton method with preconditioned conjugate gradient implemented in this package. What we intend to see is how the algorithms scale with the number of cores, so we compare the respective running time speedup of the algorithms. In this experiment, we present the results of (53) with $C = 1$, and take historical information from the latest ten iterations for L-CommDir. In addition to the datasets in Table 2, we also consider some larger ones listed in Table 6.[7] For each algorithm, the speedup is computed as:

$$\text{Speedup of using } k \text{ cores} = \frac{\text{Running time of using 1 core}}{\text{Running time of using } k \text{ cores}}.$$

In this experiment, all solvers are run on an Intel multi-core dualsocket machine with 180 GB memory. Each socket is associated with 20 cores, and we enforce all the threads to use cores from the same socket. Parallelization is achieved through openMP and Intel Math Kernel Library. The results are in Fig. 1. We see that for a9a, real-sim, and news20, L-CommDir has much better speedup than Multicore-LIBLINEAR. Moreover, L-CommDir-Diag achieves significant parallelism on epsilon, and for others the speedup of L-CommDir and Multicore-LIBLINEAR are similar. This together with the high speedup show that multicore parallelization is indeed very useful for L-CommDir to reduce the running time.

8 Distributed optimization

For distributed environments that connect multiple machines through a network, the situation is quite different. By using multiple machines, the memory bandwidth and computational power are usually less severe a problem, but often the expensive inter-

[6] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/multicore-liblinear/.

[7] All are downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

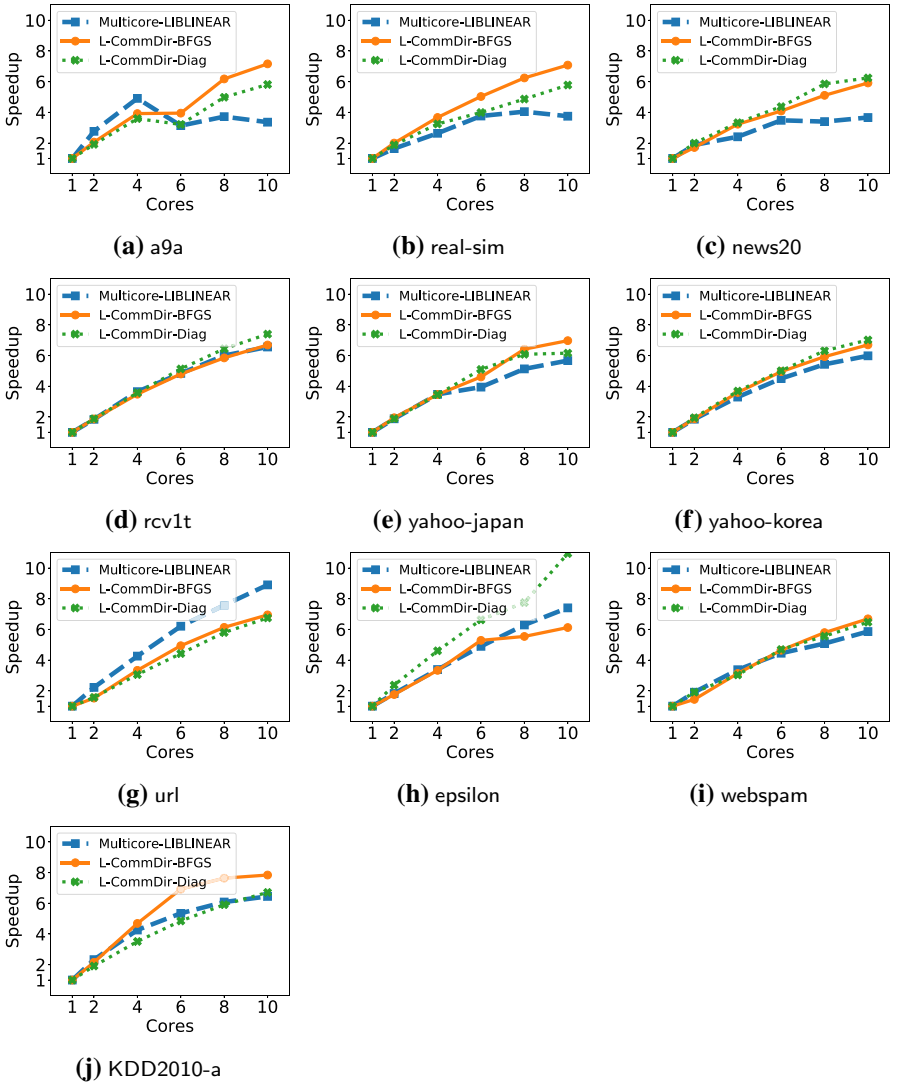

Fig. 1 Speedup using multiple cores

machine communication becomes the bottleneck. In the standard setting of distributed optimization with K machines, f is a summation of individual functions

$$f(\boldsymbol{w}) = \sum_{k=1}^{K} f_k(\boldsymbol{w}), \tag{59}$$

where each f_k is exclusively available only on the kth machine. Thus it takes one round of inter-machine communication of an n-dimensional vector to compute the gradient. For Newton method, computing the whole Hessian needs to communicate $O(n^2)$

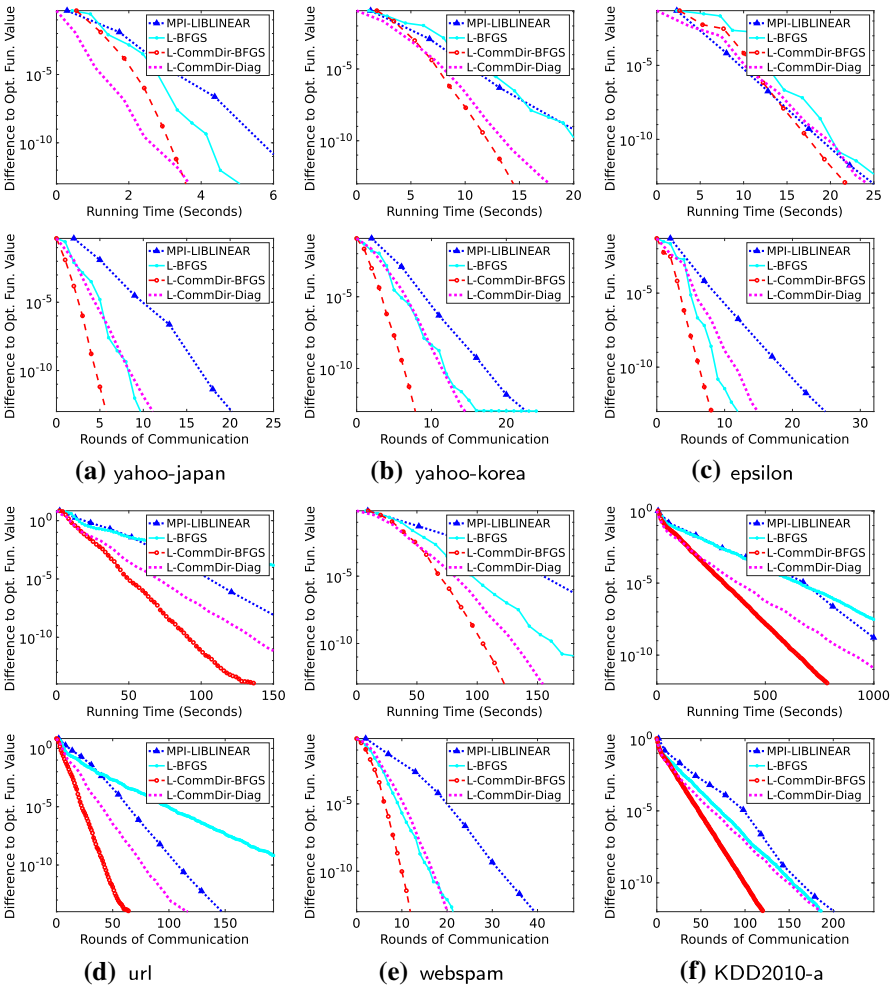

Fig. 2 Comparison of different algorithms with $C = 10^{-3}$ for (53). We show running time (left) and rounds of communication (right) v.s. (55)

elements, which is prohibitively expensive. Therefore, conjugate gradient (CG) that computes Hessian-vectors is adopted, as each Hessian-vector requires communicating an n-dimensional vector only. State of the art for distributed optimization [13, 27, 28] are Newton-CG methods with different preconditioners to reduce the overall rounds of communication needed, as Newton methods are fast-convergent asymptotically, and preconditioners could reduce the needed CG iterations per Newton step, but in practice these approaches can still take many CG iterations.

In distributed optimization, trading computation for communication could help reduce the running time. Our method that interpolates between first- and second-order methods is a perfect case for this purpose. The computation of the linear system in this setting is by letting each machine compute

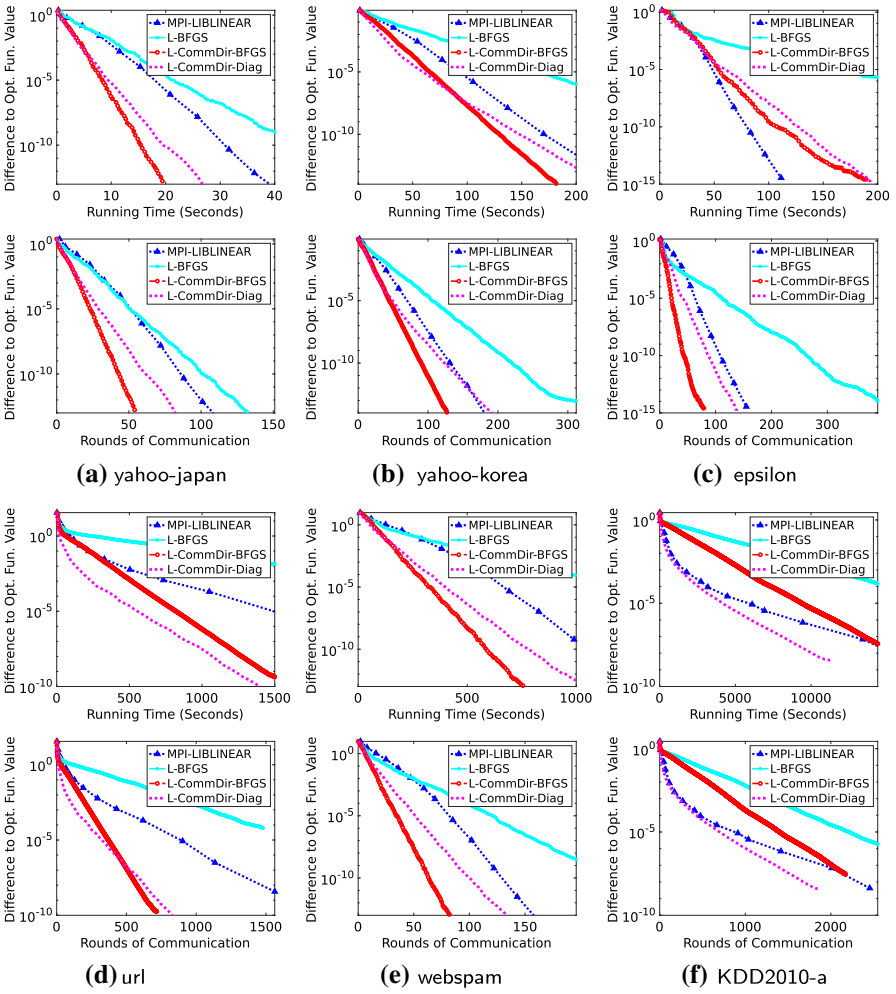

Fig. 3 Comparison of different algorithms with $C = 1$ for (53). We show running time (left) and rounds of communication (right) v.s. (55)

$$P_k^\top \nabla^2 f_k(\boldsymbol{w})\, P_k$$

locally and then use communication to do the summation over k. This involves little communication (as the matrix is just m by m) but is computation-heavy, while our method has fast empirical convergence similar to second-order methods because the real Hessian is used, making the convergence faster and therefore cutting the required rounds of communication. In comparison with first-order methods, our method is guaranteed to be no worse in terms of the worst-case communication complexity through Theorem 1, while the empirical performance is usually magnitudes better. On the other hand, in addition to the $O(n)$ communication cost for computing the

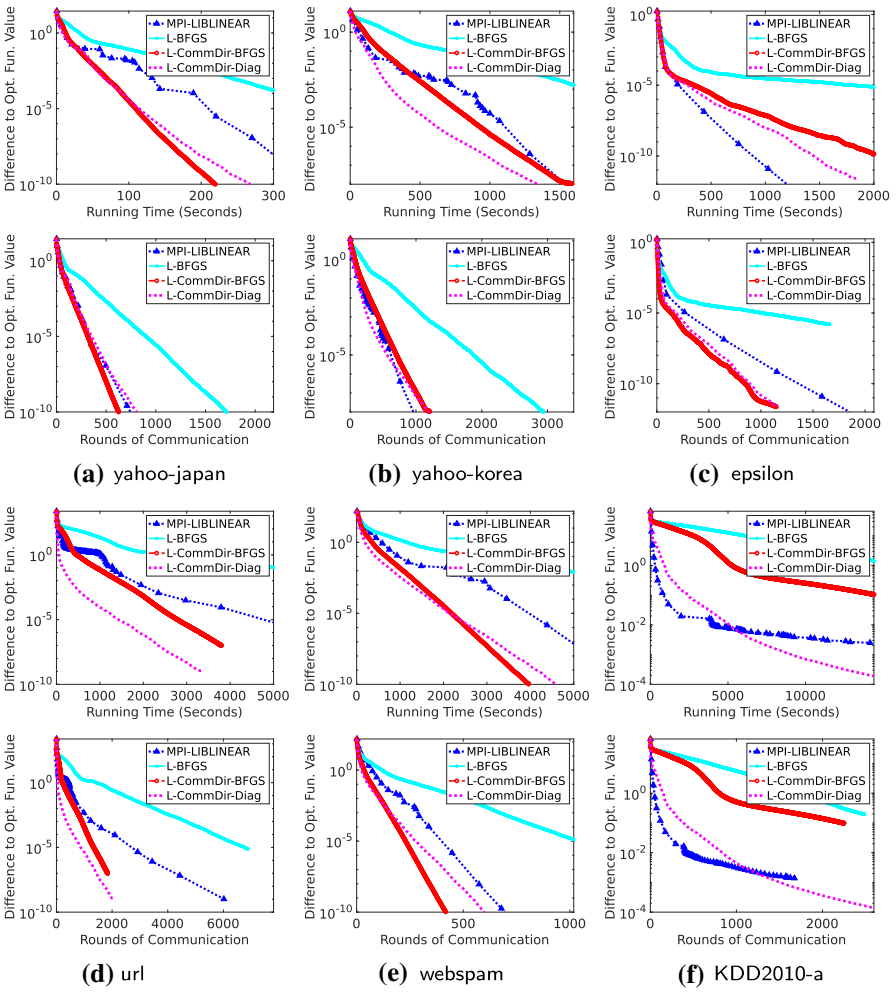

Fig. 4 Comparison of different algorithms with $C = 10^3$ for (53). We show running time (left) and rounds of communication (right) v.s. (55)

full gradient due to the distributed nature in (59), our method takes only $O(m^2)$ communication cost to obtain a non-truncated Newton step in a subspace. In summary, the computational cost per machine of Algorithm 3 in a distributed environment with K machines is the same as (58), while the communication cost per tieration is

$$O\left((m^2+n)\mathrm{Comm}(K)\right),$$

where $\mathrm{Comm}(K)$ is the communication cost of transmitting one scalar among K machines.

In comparison to Newton-CG methods that conduct multiple CG iterations per truncated Newton step, each CG iteration has communication cost $O(n)$ and our

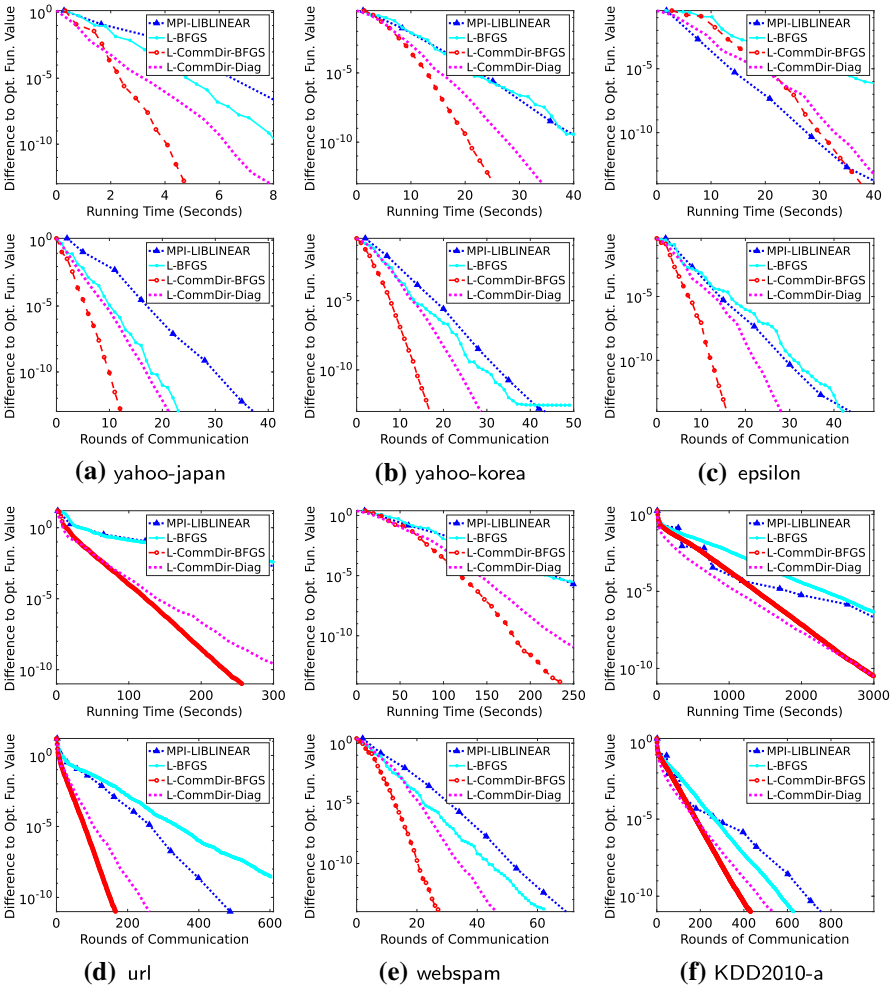

Fig. 5 Comparison of different algorithms with $C = 10^{-3}$ for (54). We show running time (left) and rounds of communication (right) v.s. (55)

obtained step is a non-truncated full Newton step in a subsapce. Thus, when the subspace is close to a solution, our method can be much more communication-efficient than Newton-CG methods.

8.1 Distributed experiments

We proceed to examine the empirical performance of our method for distributed optimization. The distributed environment is a cluster of ten machines running MPI connected through a 1Gbps network. We compare the following methods.

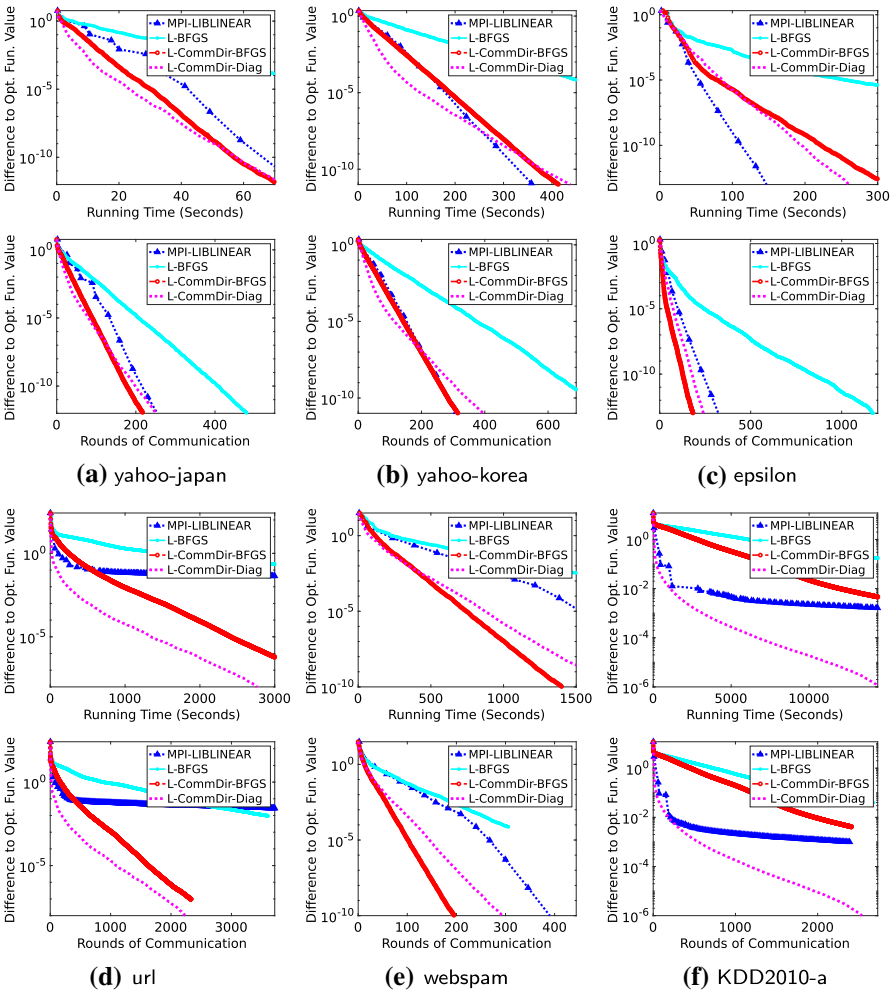

Fig. 6 Comparison of different algorithms with $C = 1$ for (54). We show running time (left) and rounds of communication (right) v.s. (55)

– MPI-LIBLINEAR:[8] A public package for distributed optimization on ERM problems. We use the distributed trust region Newton solver with preconditioned conjugate gradient using the diagonal entries of the Hessian as the preconditioner [8, 13, 28].
– VL-BFGS [6]: An implementation of L-BFGS specialized for distributed environments. We use historical information from the previous 10 iterations.
– L-CommDir: our method with (33) and (34). We use the historical information from the latest 10 iterations.

The results of $C \in \{10^{-3}, 1, 10^{3}\}$ are shown in Figs. 2, 3, 4, 5, 6 and 7.

[8] Version 2.20, downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear/.

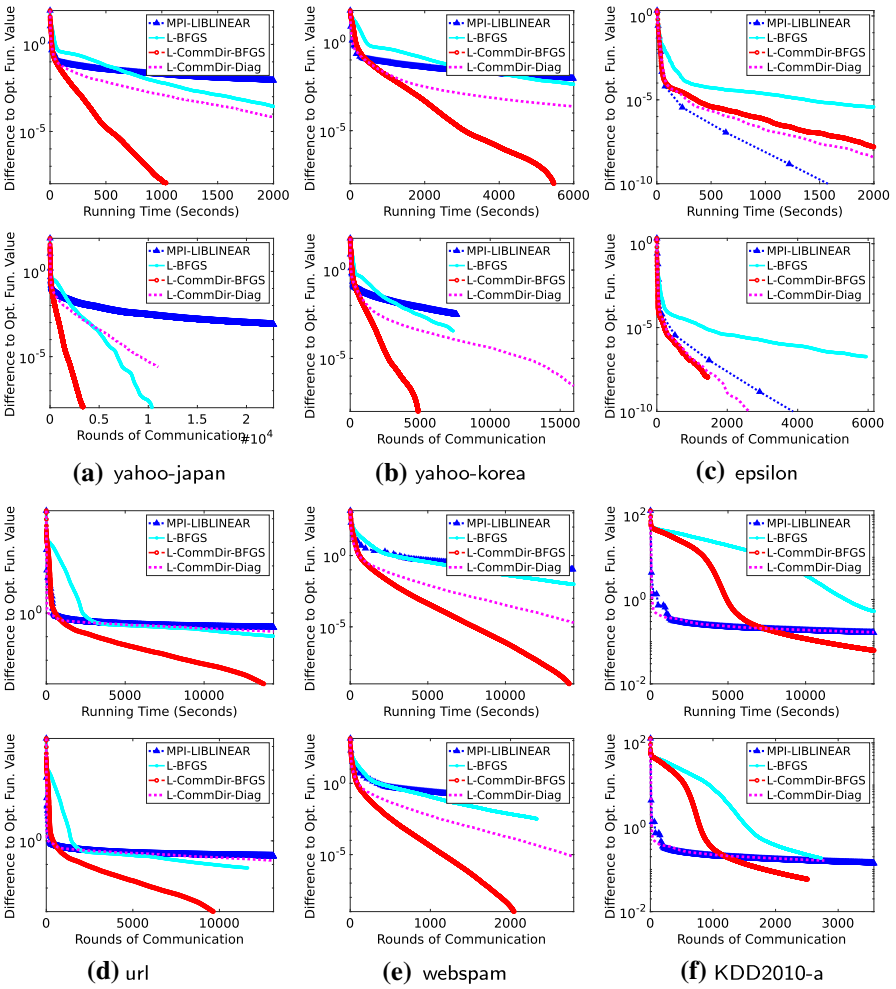

Fig. 7 Comparison of different algorithms with $C = 10^3$ for (54). We show running time (left) and rounds of communication (right) v.s. (55)

We first examine the communication cost. For (53), L-CommDir-BFGS is among the most communication-efficient, and performs worse only on KDD2010-a with $C = 1$ and $C = 10^3$. Similar trends are observed for (54), but for (54) with $C = 10^3$, L-CommDir-BFGS always outperforms other methods in communication efficiency. Likely this is because the generalized Hessian of (54) is not as useful as the real Hessian of (53), especially for difficult problems, so L-CommDir-Diag and MPI-LIBLINEAR do not perform that well. On the other hand, L-CommDir-Diag is not as communication-efficient as L-CommDir-BFGS in general, and it usually performs closer to existing methods, but for difficult problems like KDD2010-a and url, it sometimes becomes the best method in terms of the rounds of communication.

Next, regarding the running time, it and the round of communication are positively correlated, but it is also dependent on the computational power of the machines and the distributed environment. In particular, we see that L-CommDir-Diag is relatively faster in the running time, because its amount of computation per communication round is less (for every iteration it takes two communication rounds while L-CommDir-BFGS takes only one). On the other hand, MPI-LIBLINEAR is much faster than L-CommDir on epsilon, as epsilon has a lower problem dimension, making the communication cost less significant in the overall running time.

9 Conclusions

In this work, we present an efficient smooth optimization algorithm that interpolates between first- and second-order methods by utilizing information from previous iterations. Theoretical results show that our method possesses the optimal convergence rates of first-order methods while being strictly descent in the objective value. Empirical results also show that our method outperforms optimal first-order methods and second-order methods on real-world empirical risk minimization problems in single-core, multicore, and distributed optimization. Future work includes extending our method to general regularized problems by adding the regularization term to the subproblem, and to consider nonlinear manifolds in the construction of G, possibly through Riemannian optimization or partly smooth functions. Based on this work, we have expanded the package MPI-LIBLINEAR, available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear/, to include the proposed method.

Acknowledgements Parts of this wok were done when Ching-pei was in the Department of Mathematics of the National University of Singapore. The authors thank the support of HPC@NUS IT for the multicore experiment environment.

Funding This work was supported in part by MOST of Taiwan Grants 110-2221-E-002-115-MY3 and 109-2222-E-001-003-MY3.

Data availability statement Implementation of our algorithm has been incorporated as a part of the open-source package MPI-LIBLINEAR available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributedliblinear/. Code for reproducing the experiments in the paper is available at https://www.csie.ntu.edu.tw/~cjlin/papers/l-commdirlcommdir-journal-exp.tar.gz, and the datasets used in the experiments except yahoo-japan and yahoo-korea are downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted

by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

A Proofs

A.1 Proof of Theorem 1

Proof The main idea of our proof follows from the estimating sequence by [19], but we also develop techniques to cope with inexact subproblem solutions and that there is only one sequence of iterates in our algorithm. We will construct a sequence of functions $\{\phi_k\}$ such that

$$\phi_{k+1}(\boldsymbol{w}) - f(\boldsymbol{w}) \leq \prod_{i=0}^{k} (1-\alpha_i)\left(\phi_0(\boldsymbol{w}) - f(\boldsymbol{w})\right), \forall k \geq 0, \tag{60}$$

$$\phi_k^* := \min_{\boldsymbol{w}} \phi_k(\boldsymbol{w}) \geq f(\boldsymbol{w}_k) - \prod_{i=0}^{k-1} (1-\alpha_i)\,\hat{A}_{k-1}, \tag{61}$$

where

$$\hat{A}_k := \begin{cases} 0 & \text{if } \sigma > 0 \text{ or } k = 0, \\ A_k & \text{else,} \end{cases} \tag{62}$$

and show that the convergence speed $\prod_{i=0}^{k}(1-\alpha_i)$ meets the optimal rates. Let $\boldsymbol{w}^*$ be an arbitrary point in the solution set. The convergence speed of $f(\boldsymbol{w}_k) - f^*$ is then established by

$$\begin{aligned} f(\boldsymbol{w}_k) - f^* &\overset{(61)}{\leq} \phi_k^* - f^* \\ &\quad + \prod_{i=0}^{k-1} (1-\alpha_i)\,\hat{A}_{k-1} \leq \phi_k(\boldsymbol{w}^*) - f(\boldsymbol{w}^*) + \prod_{i=0}^{k-1} (1-\alpha_i)\,\hat{A}_{k-1} \\ &\overset{(60)}{\leq} \prod_{i=0}^{k-1} (1-\alpha_i)\left(\phi_0(\boldsymbol{w}^*) - f(\boldsymbol{w}^*) + \hat{A}_{k-1}\right). \end{aligned} \tag{63}$$

This sequence of functions is constructed as follows.

$$\phi_0(\boldsymbol{w}) := \frac{\gamma_0}{2}\|\boldsymbol{w} - \boldsymbol{w}_0\|^2 + f(\boldsymbol{w}_0), \tag{64}$$

$$\begin{aligned} \phi_{k+1}(\boldsymbol{w}) := (1-\alpha_k)\,\phi_k(\boldsymbol{w}) \\ + \alpha_k\left(f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{w} - \boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{w} - \boldsymbol{w}_k\|^2\right), k \geq 0. \end{aligned} \tag{65}$$

I. Upper Bound for ϕ_k:

We can show (60) easily by the definition (65) as follows.

$$\begin{aligned}\phi_{k+1}(\boldsymbol{w}) &= (1-\alpha_k)\,\phi_k(\boldsymbol{w}) + \alpha_k\left(f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{w}-\boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{w}-\boldsymbol{w}_k\|^2\right)\\ &\le (1-\alpha_k)\,\phi_k(\boldsymbol{w}) + \alpha_k f(\boldsymbol{w}),\end{aligned} \tag{66}$$

where (66) is from the (strong) convexity of f. The result (60) then follows from deducting $f(\boldsymbol{w})$ from both sides of (66) and recursion.

II. ϕ_k^* Bounds $f(\boldsymbol{w}_k)$:

Next, we prove (61). It is straightforward from induction, the definition (64)–(65), and the definition of γ_k in (8) that

$$\begin{aligned}\nabla^2\phi_k(\boldsymbol{w}) &= (1-\alpha_{k-1})\nabla^2\phi_{k-1}(\boldsymbol{w}) + \alpha_{k-1}\sigma I\\ &= (1-\alpha_{k-1})\gamma_{k-1}I + \alpha_{k-1}\sigma I = \gamma_k I, \quad \forall k,\end{aligned}$$

showing that ϕ_k are quadratic and strongly convex with modulus γ_k. Therefore, at the kth iteration, there is an optimal solution $\boldsymbol{u}_k$ such that

$$\phi_k(\boldsymbol{w}) = \phi_k^* + \frac{\gamma_k}{2}\|\boldsymbol{w}-\boldsymbol{u}_k\|^2, \ \forall \boldsymbol{w}. \tag{67}$$

By setting the derivative of (65) to zero and using (67), we get that

$$\begin{aligned}\mathbf{0} &= \nabla\phi_{k+1}(\boldsymbol{u}_{k+1}) = (1-\alpha_k)\,\nabla\phi_k(\boldsymbol{u}_{k+1}) + \alpha_k\left(\nabla f(\boldsymbol{w}_k) + \sigma(\boldsymbol{u}_{k+1}-\boldsymbol{w}_k)\right)\\ &= (1-\alpha_k)\,\gamma_k(\boldsymbol{u}_{k+1}-\boldsymbol{u}_k) + \alpha_k\left(\nabla f(\boldsymbol{w}_k) + \sigma(\boldsymbol{u}_{k+1}-\boldsymbol{w}_k)\right).\end{aligned}$$

Rearranging the above equation, we have

$$\boldsymbol{u}_{k+1} = \frac{1}{(1-\alpha_k)\gamma_k + \alpha_k\sigma}\left((1-\alpha_k)\gamma_k\boldsymbol{u}_k + \alpha_k\sigma\boldsymbol{w}_k - \alpha_k\nabla f(\boldsymbol{w}_k)\right),$$

which satisfies the definition of $\boldsymbol{v}_k$ in (7) and (8) (notice that $(1-\alpha_k)\gamma_k + \alpha_k\sigma > 0$ as discussed when we define (7) so $\boldsymbol{u}_{k+1}$ is well-defined). Thus, we have

$$\boldsymbol{u}_k \equiv \boldsymbol{v}_k, \quad \text{and} \quad \phi_k^* = \phi_k(\boldsymbol{v}_k), \forall k \ge 0, \tag{68}$$

for both cases of $\sigma > 0$ and $\sigma = 0$, and (67) can be written as

$$\phi_k(\boldsymbol{w}) = \phi_k(\boldsymbol{v}_k) + \frac{\gamma_k}{2}\|\boldsymbol{w}-\boldsymbol{v}_k\|^2, \forall k \ge 0. \tag{69}$$

First consider the case of $\sigma > 0$. From (7) and (10), we have

$$\boldsymbol{v}_{k+1} = (1-\alpha)\boldsymbol{v}_k + \alpha\boldsymbol{w}_k - \frac{\alpha}{\sigma}\nabla f(\boldsymbol{w}_k),$$

which can be written as the following two equivalent forms.

$$\boldsymbol{v}_{k+1} - \boldsymbol{v}_k = -\alpha(\boldsymbol{v}_k - \boldsymbol{w}_k) - \frac{\alpha}{\sigma}\nabla f(\boldsymbol{w}_k), \tag{70}$$

$$\boldsymbol{v}_{k+1} - \boldsymbol{w}_k = (1-\alpha)(\boldsymbol{v}_k - \boldsymbol{w}_k) - \frac{\alpha}{\sigma}\nabla f(\boldsymbol{w}_k). \tag{71}$$

We then obtain a recurrent relation of ϕ^*_{k+1} by using (69), (10), and (65).

$$\begin{aligned}
\phi^*_{k+1} &= \phi_{k+1}(\boldsymbol{v}_{k+1}) \\
&= (1-\alpha)\left(\phi_k(\boldsymbol{v}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_{k+1} - \boldsymbol{v}_k\|^2\right) \\
&\quad + \alpha\left(f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_{k+1} - \boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_{k+1} - \boldsymbol{w}_k\|^2\right) && (72) \\
&= (1-\alpha)\phi_k(\boldsymbol{v}_k) + \alpha f(\boldsymbol{w}_k) && (73) \\
&\quad + \alpha(1-\alpha)\left(\nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_k - \boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_k - \boldsymbol{w}_k\|^2\right) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2, && (74)
\end{aligned}$$

where (72) is by (65), (10), and (69); and (73)–(74) are obtained from separately applying (70)–(71), and applying (10) to the coefficient of $\|\nabla f(\boldsymbol{w}_k)\|^2$. Now with (68) and the definition that $\hat{A}_k = 0,\ \forall k \geq 0$, we are ready to prove an equivalent form of (61), $\phi_k(\boldsymbol{v}_k) \geq f(\boldsymbol{w}_k)$, $\forall k$, by induction. From the definition (64), it is clear that $\phi_0(\boldsymbol{v}_0) \geq f(\boldsymbol{w}_0)$. For the induction, assume $\phi_k(\boldsymbol{v}_k) \geq f(\boldsymbol{w}_k)$ for some $k \geq 0$. For the next $(k+1)$th iteration, replacing the first term in (73) with $f(\boldsymbol{w}_k)$ leads to

$$\phi^*_{k+1} \geq f(\boldsymbol{w}_k) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2 + \alpha(1-\alpha)\left(\nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_k - \boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_k - \boldsymbol{w}_k\|^2\right),$$

which together with (15) leads to $\phi_{k+1}(\boldsymbol{v}_{k+1}) \geq f(\boldsymbol{w}_{k+1})$. Thus the bound (61) holds.

Next, we consider the case of $\sigma = 0$. From the recursion (8), we have

$$\gamma_{k+1} = \rho\prod_{i=0}^{k}(1-\alpha_i), \quad \forall k \geq 0. \tag{75}$$

Further, in this case, (7) is reduced to

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k - \frac{\alpha_k}{\gamma_{k+1}}\nabla f(\boldsymbol{w}_k). \tag{76}$$

We obtain the following recurrent relation of ϕ^*_{k+1}.

$$\begin{aligned}
&\phi_{k+1}(\boldsymbol{v}_{k+1}) \\
&\overset{(65),(69)}{=} (1-\alpha_k)\left(\phi_k(\boldsymbol{v}_k) + \frac{\gamma_k}{2}\|\boldsymbol{v}_{k+1} - \boldsymbol{v}_k\|^2\right) + \alpha_k\left(f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_{k+1} - \boldsymbol{w}_k)\right)
\end{aligned} \tag{77}$$

Note that

$$\frac{1}{2}(1-\alpha_k)\gamma_k \|\boldsymbol{v}_{k+1}-\boldsymbol{v}_k\|^2 \overset{(76)}{=} \frac{1}{2}(1-\alpha_k)\gamma_k \cdot \frac{\alpha_k^2}{\gamma_{k+1}^2}\|\nabla f(\boldsymbol{w}_k)\|^2 \overset{(8)}{=} \frac{\alpha_k^2}{2\gamma_{k+1}}\|\nabla f(\boldsymbol{w}_k)\|^2.$$

Applying the above result to (77), we have

$$\begin{aligned}\phi_{k+1}(\boldsymbol{v}_{k+1}) &\overset{(76)}{=} (1-\alpha_k)\phi_k(\boldsymbol{v}_k) + \alpha_k f(\boldsymbol{w}_k) + \frac{\alpha_k^2}{2\gamma_{k+1}}\|\nabla f(\boldsymbol{w}_k)\|^2 \\ &\quad + \alpha_k \nabla f(\boldsymbol{w}_k)^\top \left(\boldsymbol{v}_k - \boldsymbol{w}_k - \frac{\alpha_k}{\gamma_{k+1}}\nabla f(\boldsymbol{w}_k)\right) \\ &\overset{(11)}{=} (1-\alpha_k)\phi_k^* + \alpha_k f(\boldsymbol{w}_k) - \frac{\|\nabla f(\boldsymbol{w}_k)\|^2}{2\rho} + \alpha_k \nabla f(\boldsymbol{w}_k)^\top (\boldsymbol{v}_k - \boldsymbol{w}_k).\end{aligned} \tag{78}$$

We know that $\phi_0(\boldsymbol{v}_0) \geq f(\boldsymbol{w}_0)$ holds from the construction (64). Now we assume that (61) holds for some $k \geq 0$ and obtain for the $(k+1)$th iteration that

$$\begin{aligned}\phi_{k+1}^* &\overset{(78),(61)}{\geq} f(\boldsymbol{w}_k) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2 - (1-\alpha_k)\prod_{i=0}^{k-1}(1-\alpha_i)A_{k-1} + \alpha_k \nabla f(\boldsymbol{w}_k)^T(\boldsymbol{v}_k - \boldsymbol{w}_k) \\ &\overset{(17)}{\geq} f(\boldsymbol{w}_k) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2 - (1-\alpha_k)\prod_{i=0}^{k-1}(1-\alpha_i)A_{k-1} - \alpha_k\psi_k \\ &\overset{(17)}{\geq} f(\boldsymbol{w}_{k+1}) - (1-\alpha_k)\prod_{i=0}^{k-1}(1-\alpha_i)A_{k-1} - \alpha_k\psi_k \\ &\overset{(75),(11)}{\geq} f(\boldsymbol{w}_{k+1}) - \prod_{i=0}^{k}(1-\alpha_i)\left(A_{k-1} + \frac{\psi_k}{\alpha_k}\right) \\ &\overset{(19)}{=} f(\boldsymbol{w}_{k+1}) - \prod_{i=0}^{k}(1-\alpha_i)A_k.\end{aligned}$$

Thus we have shown by induction that (61) holds for all $k \geq 0$.

III. Convergence Speed of $\phi_k(\boldsymbol{w}^*) - f(\boldsymbol{w}^*)$:

We proceed to show how fast $\prod_{i=0}^{k}(1-\alpha_i)$ converges to zero. First consider the case that $\sigma > 0$. Clearly, (10) shows that the term $\prod_{i=0}^{k}(1-\alpha_i)$ converges to zero at the desired optimal linear convergence rate of $(1-\sqrt{\sigma/\rho})$. With (63), $\hat{A}_k = 0,\ \forall k \geq 0$, and replacing $\phi_0(\boldsymbol{w}^*)$ with the right-hand side of (64) and γ_0 with σ via (8), this implies the optimal rate for first-order methods on strongly convex problems shown in (16), and hence the desired iteration complexity.

When $\sigma = 0$, from that $\alpha_k > 0$ and the fact that $\sqrt{a^2+b^2} \leq |a| + |b|$, we get

$$\alpha_{k+1} = \frac{-\alpha_k^2 + \alpha_k\sqrt{\alpha_k^2+4}}{2} \leq \frac{-\alpha_k^2 + \alpha_k(\alpha_k+2)}{2} = \alpha_k, \quad \forall k \geq 0.$$

Thus, the sequence of α_k is decreasing. Further, we have $\alpha_k \in (0, 1)$ for all k from the decreasing property and that

$$0 = \frac{-1+1}{2} < \alpha_0 = \frac{-1+\sqrt{5}}{2} < \frac{-1+\sqrt{9}}{2} = 1.$$

Let us define $\lambda_k := \alpha_k^{-1}$. From the decreasing property of $\{\alpha_k\}$ and (12), we have that

$$\lambda_{k+1} - \lambda_k = \frac{\alpha_k - \alpha_{k+1}}{\alpha_k \alpha_{k+1}} = \frac{\alpha_k^2 - \alpha_{k+1}^2}{\alpha_k \alpha_{k+1}\left(\alpha_k + \alpha_{k+1}\right)} \geq \frac{\alpha_k^2 - \alpha_{k+1}^2}{2\alpha_k^2 \alpha_{k+1}} = \frac{\alpha_k^2(1-(1-\alpha_{k+1}))}{2\alpha_k^2 \alpha_{k+1}} = \frac{1}{2}.$$

Therefore, $\lambda_k \geq k/2 + \lambda_0$. This result, (13), and the definition of λ_k imply

$$\alpha_k^{-1} \geq \frac{k}{2} + \alpha_0^{-1} = \frac{k}{2} + \frac{2}{\sqrt{5}-1} = \frac{k}{2} + \frac{2\left(\sqrt{5}+1\right)}{4} \geq \frac{k+2}{2}, \forall k \geq 0. \quad (79)$$

We therefore get that from (75), (11) and (79),

$$\rho \prod_{i=0}^{k} (1-\alpha_i) = \gamma_{k+1} = \rho \alpha_k^2 \leq \frac{4\rho}{(k+2)^2}, \quad \forall k \geq 0, \quad (80)$$

satisfying the optimal convergence speed of $O(k^{-2})$. The form (18) is then obtained by first inserting (64) and (80) into (63) to get

$$f(\boldsymbol{w}_k) - f^* \leq \frac{4}{(k+1)^2}\left(\frac{\gamma_0}{2}\left\|\boldsymbol{w}_0 - \boldsymbol{w}^*\right\|^2 + f(\boldsymbol{w}_0) - f^* + \hat{A}_{k-1}\right),$$

and then replacing γ_0 with ρ by (8) and $\hat{A}_k$ with the values in (62) and (19). □

A.2 Proof of Proposition 1

Proof From the strong convexity of f, even though there might be multiple possibilities of $\boldsymbol{t}$ that are optimal for (2), they all result in the same iterate $\boldsymbol{w}_{k+1}^*$ after the mapping $\boldsymbol{w}_k + G(\boldsymbol{t})$. We first show that (21) holds at any optimal $\boldsymbol{t}^*$ that maps to $\boldsymbol{w}_{k+1}^*$ and then discuss its neighborhood. As G is linear, we have that there is $P \in \mathbf{R}^{n \times m}$ such that

$$G(\boldsymbol{t}) = P\boldsymbol{t}, \quad \forall \boldsymbol{t} \in \mathbf{R}^m.$$

Therefore, the optimality condition of the convex problem (2) gives

$$\left.\frac{\partial f(\boldsymbol{w}_k + G(\boldsymbol{t}))}{\partial \boldsymbol{t}^\top}\right|_{\boldsymbol{t}=\boldsymbol{t}^*} = \nabla f(\boldsymbol{w}_{k+1}^*)^\top P = \mathbf{0}^\top,$$

for any $\boldsymbol{t}^*$ that is optimal for (2), and thus

$$\nabla f(\boldsymbol{w}_{k+1}^*)^\top G\left(\boldsymbol{t}^*\right) = \nabla f(\boldsymbol{w}_{k+1}^*)^\top P\boldsymbol{t}^* = 0, \tag{81}$$

$$\nabla f(\boldsymbol{w}_{k+1}^*)^\top \tilde{\boldsymbol{v}}_{k+1} = 0, \tag{82}$$

where (82) holds because the assumption $\tilde{\boldsymbol{v}}_{k+1} \in \mathtt{span}\left(\nabla f(\boldsymbol{w}_k), \tilde{\boldsymbol{v}}_{k+1}\right) \subseteq \mathtt{Im}(G)$ implies the existence of some $\hat{\boldsymbol{t}}$ such that $\tilde{\boldsymbol{v}}_{k+1} = P\hat{\boldsymbol{t}}$. By combining (82) and (81), we get

$$\nabla f\left(\boldsymbol{w}_k + G\left(\boldsymbol{t}^*\right)\right)^\top \left(\tilde{\boldsymbol{v}}_{k+1} - G\left(\boldsymbol{t}^*\right)\right) = 0, \tag{83}$$

so (21a) holds at $\boldsymbol{w}_{k+1}^*$ because the first term is zero and the second term is nonnegative. For (21b), it is clear from the Lipschitz continuity of the gradient of f that

$$\begin{aligned} f\left(\boldsymbol{w}_{k+1}^*\right) &\le f\left(\boldsymbol{w}_k - \frac{1}{\rho}\nabla f\left(\boldsymbol{w}_k\right)\right) \\ &\le f\left(\boldsymbol{w}_k\right) - \nabla f\left(\boldsymbol{w}_k\right)^\top \left(\frac{1}{\rho}\nabla f\left(\boldsymbol{w}_k\right)\right) + \frac{\rho}{2}\left\|\frac{1}{\rho}\nabla f\left(\boldsymbol{w}_k\right)\right\|^2 \\ &= f\left(\boldsymbol{w}_k\right) - \frac{1}{2\rho}\left\|\nabla f\left(\boldsymbol{w}_k\right)\right\|^2. \end{aligned} \tag{84}$$

If $\boldsymbol{w}_{k+1}^* = \boldsymbol{w}_k + G(\boldsymbol{t}^*)$ lies in the interior of (21) (i.e., $\boldsymbol{w}_{k+1}^*$ strictly satisfies the inequalities in (21)), by the continuity of f and ∇f, there is a neighborhood of $\boldsymbol{w}_{k+1}^*$ such that every point of it satisfies (21). As G is also continuous, it means that for each optimal solution $\boldsymbol{t}^*$ there is a neighborhood of which all points satisfy (21).

Now we discuss the cases when $\boldsymbol{w}_{k+1}^*$ lies on the boundary of (21). When (21a) holds in equality at $\boldsymbol{w}_{k+1}^*$, (82) and (81) from the optimality condition at $\boldsymbol{w}_{k+1}^*$ imply that

$$\tilde{\boldsymbol{v}}_{k+1} = G(\boldsymbol{t}^*).$$

When (21b) holds in equality at $\boldsymbol{w}_{k+1}^* = \boldsymbol{w}_k + G(\boldsymbol{t}^*)$, (84) implies that

$$f\left(\boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k)\right) = f\left(\boldsymbol{w}_{k+1}^*\right). \tag{85}$$

By the assumption that $\nabla f(\boldsymbol{w}_k) \in \mathtt{Im}(G)$, $\boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k)$ is in the form of $\boldsymbol{w}_k + G(\boldsymbol{t})$. We have mentioned in the beginning of the proof that the optimal $\boldsymbol{w}_k + G(\boldsymbol{t}^*)$ for (2) is unique. Thus (85) implies that

$$\boldsymbol{w}_{k+1}^* = \boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k).$$

□

A.3 Proof of Proposition 2

Proof We note that (83) holds at any optimum $\boldsymbol{t}^*$ as its required conditions are all satisfied. Therefore, (22a) holds within a neighborhood of any optimal solution of (2) as long as $\psi_{k+1} > 0$ since the inequality in (22a) is strictly satisfied at $\boldsymbol{w}_{k+1}^*$. The part for (22b) is similar to the argument in the proof of Proposition 1. Notice that when (22b) holds at equality, $G(\boldsymbol{t}^*) = -\nabla f(\boldsymbol{w}_k)/\rho$ is an optimal solution and thus (83) holds at $\boldsymbol{t}^*$ as well. Then with $\psi_{k+1} > 0$ the inequality in (22a) strictly holds. □

A.4 Proof of Theorem 2

Proof The solution of (25) also solves the following linear system.

$$P_k^\top H_k P_k \boldsymbol{t}_k = -P_k^\top \nabla f(\boldsymbol{w}_k), \tag{86}$$

where we recall that P_k is defined in (27). If $\boldsymbol{q}_j^k$ satisfies (28), then the right-hand side of (86) is not all zero, hence $\boldsymbol{t}_k \neq \boldsymbol{0}$ and $P_k \boldsymbol{t}_k \neq \boldsymbol{0}$. Therefore, from (86), we have

$$-\boldsymbol{p}_k^\top \nabla f(\boldsymbol{w}_k) = (P_k \boldsymbol{t}_k)^\top H_k P_k \boldsymbol{t}_k \geq M_2 \|P_k \boldsymbol{t}\|^2 = M_2 \|\boldsymbol{p}_k\|^2. \tag{87}$$

We then have from Assumption 1 and (87) that

$$\begin{aligned} f(\boldsymbol{w}_k + \theta_k \boldsymbol{p}_k) &\leq f(\boldsymbol{w}_k) + \theta_k \nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k + \theta_k^2 \frac{\rho}{2} \|\boldsymbol{p}_k\|^2 \\ &\leq f(\boldsymbol{w}_k) + \theta_k \nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k \left(1 - \frac{\rho \theta_k}{2M_2}\right). \end{aligned}$$

From (87), $\nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k < 0$. Therefore, when

$$1 - \frac{\rho \theta_k}{2M_2} \geq c_1,$$

(24) is satisfied. Thus, by considering the possibility of overshoot, we obtain a lower bound $\bar{\theta}$ of the final step size in (29), and the backtracking procedure takes at most $\lceil \log_\beta \bar{\theta} \rceil$ steps. □

A.5 Proof of Theorem 3

Proof The jth equation in the linear system (86) is

$$\boldsymbol{p}_k^\top H_k \boldsymbol{q}_j^k = -\nabla f(\boldsymbol{w}_k)^\top \boldsymbol{q}_j^k. \tag{88}$$

By (26), (28), and (88),

$$\|\boldsymbol{p}_k\| \|\boldsymbol{q}_j^k\| \geq \left| \frac{1}{M_1} (\boldsymbol{p}_k)^\top H_k \boldsymbol{q}_j^k \right| = \left| \frac{1}{M_1} \nabla f(\boldsymbol{w}_k)^\top \boldsymbol{q}_j^k \right| \geq \|\nabla f(\boldsymbol{w}_k)\| \|\boldsymbol{q}_j^k\| \frac{\delta}{M_1}.$$

Therefore,

$$\|\boldsymbol{p}_k\| \geq \frac{\delta}{M_1}\|\nabla f(\boldsymbol{w}_k)\|. \tag{89}$$

Combining (87) and (89), we can establish the following result.

$$-\frac{\boldsymbol{p}_k^\top \nabla f(\boldsymbol{w}_k)}{\|\boldsymbol{p}_k\|\|\nabla f(\boldsymbol{w}_k)\|} \geq \frac{M_2\|\boldsymbol{p}_k\|^2}{\|\boldsymbol{p}_k\|\|\nabla f(\boldsymbol{w}_k)\|} \geq \frac{\delta M_2}{M_1}.$$

By (24), (87), and (89), we have

$$f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}_k) \leq \theta_k c_1 \nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k \leq -\frac{M_2\delta^2\theta_k c_1}{M_1^2}\|\nabla f(\boldsymbol{w}_k)\|^2. \tag{90}$$

Summing (90) up from 0 to k, we get

$$\sum_{j=0}^{k} \frac{M_2\delta^2 c_1 \min_{0\leq j\leq k}\theta_j}{M_1^2}\|\nabla f(\boldsymbol{w}_j)\|^2 \leq f(\boldsymbol{w}_0) - f(\boldsymbol{w}_{k+1}) \leq f(\boldsymbol{w}_0) - f^*,$$

where f^* is the minimal objective value of f. Consequently, by (29),

$$\begin{aligned} g_k := \min_{0\leq J\leq k}\|\nabla f(\boldsymbol{w}_J)\|^2 &\leq \frac{1}{k+1}\sum_{J=0}^{k}\|\nabla f(\boldsymbol{w}_J)\|^2 \\ &\leq \frac{1}{k+1}\frac{M_1^2}{M_2\delta^2 c_1 \min_{0\leq J\leq k}\theta_J}(f(\boldsymbol{w}_0) - f^*) \\ &\leq \frac{1}{k+1}\frac{M_1^2}{M_2\delta^2 c_1\bar{\theta}}(f(\boldsymbol{w}_0) - f^*). \end{aligned}$$

Finally, the $o(1/k)$ convergence follows from that g_k is a decreasing, summable, and nonnegative sequence so Proposition 3.4 and Theorem 3.5 of [24] apply directly. Note that since

$$\sum_{J=0}^{k}\|\nabla f(\boldsymbol{w}_J)\|^2$$

is bounded, $\|\nabla f(\boldsymbol{w}_k)\|$ converges to zero as k approaches infinity. □

A.6 Proof of Theorem 4

Proof From convexity of f, we have that for any $\boldsymbol{w}^* \in \Omega$,

$$f(\boldsymbol{w}) - f^* \leq \nabla f(\boldsymbol{w})\left(\boldsymbol{w} - \boldsymbol{w}^*\right) \leq \|\nabla f(\boldsymbol{w})\|\|\boldsymbol{w} - \boldsymbol{w}^*\|.$$

From (30), we have that

$$f(\boldsymbol{w}) - f^* \leq \|\nabla f(\boldsymbol{w})\| R_0. \tag{91}$$

Define

$$\Delta_k := f(\boldsymbol{w}_k) - f^* \tag{92}$$

and therefore from (91),

$$-\|\nabla f(\boldsymbol{w}_k)\| \leq \frac{-(f(\boldsymbol{w}_k) - f^*)}{R_0} = \frac{-\Delta_k}{R_0}. \tag{93}$$

By substituting (92) into (90) and using (93), we get

$$\Delta_{k+1} \leq \Delta_k - \frac{M_2 \delta^2 \theta_k c_1}{M_1^2 R_0^2} \Delta_k^2. \tag{94}$$

Now divide (94) by $\Delta_k \Delta_{k+1}$ and note from (90) that Δ_k is monotonically decreasing. We get

$$\frac{1}{\Delta_k} \leq \frac{1}{\Delta_{k+1}} - \frac{M_2 \delta^2 \theta_k c_1}{M_1^2 R_0^2}. \tag{95}$$

By summing (95) from 0 to k, we obtain

$$\frac{1}{\Delta_0} \leq \frac{1}{\Delta_{k+1}} - \frac{M_2 \delta^2 c_1}{M_1^2 R_0^2} \sum_{j=0}^{k} \theta_j. \tag{96}$$

This together with (29) leads to

$$\Delta_{k+1} \leq \frac{\Delta_0 M_1^2 R_0^2}{M_1^2 R_0^2 + \Delta_0 M_2 \delta^2 c_1 \sum_{j=0}^{k} \theta_j} \leq \frac{\Delta_0 M_1^2 R_0^2}{M_1^2 R_0^2 + \Delta_0 M_2 \delta^2 c_1 (k+1) \bar{\theta}}. \tag{97}$$

From Assumption 1, f is ρ-Lipschitz-continuously differentiable, and from (92) and (30) we have

$$\Delta_0 = f(\boldsymbol{w}_0) - f^* \leq \frac{\rho}{2} \|\boldsymbol{w}_0 - \bar{\boldsymbol{w}}\|^2 \leq \frac{\rho}{2} R_0^2, \tag{98}$$

where

$$\bar{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}^* \in \Omega} \left\| \boldsymbol{w}_0 - \boldsymbol{w}^* \right\|, \tag{99}$$

and R_0 is finite. By noting that $\Delta_0 \geq 0$ and that the differential of the right-hand side of (97) with respect to Δ_0 is nonnegative when $\Delta_0 \geq 0$, we can use (98) to simplify (97) to

$$\Delta_k \leq \frac{\rho M_1^2 R_0^2}{2M_1^2 + \rho M_2 \delta^2 c_1 (k+1)\bar{\theta}},$$

showing the desired $O(1/k)$ rate. □

A.7 Proof of Theorem 5

Proof Deducting f^* from both sides of (90) and combining it with (32), we get that for all k

$$\begin{aligned} f(\boldsymbol{w}_{k+1}) - f^* &\leq \left(1 - \frac{2\sigma M_2 \delta^2 \theta_k c_1}{M_1^2}\right)\left(f(\boldsymbol{w}_k) - f^*\right) \\ &\leq \left(1 - \frac{2\sigma M_2 \delta^2 \bar{\theta} c_1}{M_1^2}\right)\left(f(\boldsymbol{w}_k) - f^*\right), \end{aligned} \tag{100}$$

where the last inequality is from (29). Thus, we see that (100) gives a convergence rate that is at least Q-linear. Note that our assumptions give $c_1 > 0$, and $\sigma M_2 / M_1^2 > 0$. Therefore the coefficient in the right-hand side of (100) is smaller than 1. □

References

1. Attouch, H., Peypouquet, J.: The rate of convergence of Nesterov's accelerated forward–backward method is actually faster than 1/k^2. SIAM J. Optim. **26**(3), 1824–1834 (2016)
2. Barzilai, J., Borwein, J.M.: Two-point step size gradient methods. IMA J. Numer. Anal. **8**, 141–148 (1988)
3. Bertsekas, D.P.: Nonlinear Programming, 3rd edn. Athena Scientific, Belmont (2016)
4. Burachik, R., Graña Drummond, L., Iusem, A.N., Svaiter, B.: Full convergence of the steepest descent method with inexact line searches. Optimization **32**(2), 137–146 (1995)
5. Chambolle, A., Dossal, C.: On the convergence of the iterates of "FISTA". J. Optim. Theory Appl. **166**(3), 25 (2015)
6. Chen, W., Wang, Z., Zhou, J.: Large-scale L-BFGS using MapReduce. In: Advances in Neural Information Processing Systems, pp. 1332–1340 (2014)
7. Hiriart-Urruty, J.B., Strodiot, J.J., Nguyen, V.H.: Generalized Hessian matrix and second-order optimality conditions for problems with $C^{1,1}$ data. Appl. Math. Optim. **11**(1), 43–56 (1984)
8. Hsia, C.Y., Chiang, W.L., Lin, C.J.: Preconditioned conjugate gradient methods in truncated Newton frameworks for large-scale linear classification. In: Proceedings of the Asian Conference on Machine Learning (ACML) (2018). http://www.csie.ntu.edu.tw/~cjlin/papers/tron_pcg/precondition.pdf
9. Karimi, H., Nutini, J., Schmidt, M.: Linear convergence of gradient and proximal-gradient methods under Polyak-łojasiewicz condition. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2016)
10. Lee, C.P., Wang, P.W., Chen, W., Lin, C.J.: Limited-memory common-directions method for distributed optimization and its application on empirical risk minimization. In: Proceedings of SIAM International Conference on Data Mining (SDM) (2017). http://www.csie.ntu.edu.tw/~cjlin/papers/l-commdir/l-commdir.pdf

11. LEE, C.P., Wright, S.J.: First-order algorithms converge faster than $O(1/k)$ on convex problems. In: Proceedings of the 36th International Conference on Machine Learning (ICML) (2019). http://proceedings.mlr.press/v97/lee19e.html
12. Lee, M.C., Chiang, W.L., Lin, C.J.: Fast matrix-vector multiplications for large-scale logistic regression on shared-memory systems. In: Proceedings of the IEEE International Conference on Data Mining (ICDM) (2015). http://www.csie.ntu.edu.tw/~cjlin/papers/multicore_liblinear_icdm.pdf
13. Lin, C.Y., Tsai, C.H., Lee, C., Lin, C.J.: Large-scale logistic regression and linear support vector machines using Spark. In: Proceedings of the IEEE International Conference on Big Data, pp. 519–528 (2014). http://www.csie.ntu.edu.tw/~cjlin/papers/spark-liblinear/spark-liblinear.pdf
14. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Program. **45**(1), 503–528 (1989)
15. Łojasiewicz, S.: Une propriété topologique des sous-ensembles analytiques réels. In: Les Équations aus Dérivées Partielles. Éditions du centre National de la Recherche Scientifique (1963)
16. Mangasarian, O.L.: A finite Newton method for classification. Optim. Methods Softw. **17**(5), 913–929 (2002)
17. Nemirovsky, A.S., Yudin, D.B.: Problem complexity and method efficiency in optimization. In: Wiley-Interscience Series in Discrete Mathematics. Wiley, New York (1983)
18. Nesterov, Y.E.: A method of solving a convex programming problem with convergence rate $o(1/k^2)$. Sov. Math. Doklady **27**, 372–376 (1983)
19. Nesterov, Y.E.: Introductory Lectures on Convex Optimization: A Basic Course. Kluwer (2003)
20. Nesterov, Y.E.: Gradient methods for minimizing composite functions. Math. Program. **140**(1), 125–161 (2013)
21. Nocedal, J., Wright, S.: Numerical Optimization, 2nd edn. Springer (2006)
22. Polyak, B.T.: Gradient methods for minimizing functionals. Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki **3**(4), 643–653 (1963)
23. Salzo, S., Villa, S.: Parallel random block-coordinate forward–backward algorithm: a unified convergence analysis. Math. Program. **66**, 1–45 (2021)
24. Shi, W., Ling, Q., Wu, G., Yin, W.: Extra: an exact first-order algorithm for decentralized consensus optimization. SIAM J. Optim. **25**(2), 944–966 (2015)
25. Tseng, P.: On Accelerated Proximal Gradient Methods for Convex–Concave Optimization. Department of Mathematics, University of Washington, Tech. rep. (2008)
26. Wang, P.W., Lee, C., Lin, C.J.: The common-directions method for regularized empirical risk minimization. J. Mach. Learn. Res. **20**(58), 1–49 (2019)
27. Zhang, Y., Xiao, L.: DiSCO: distributed optimization for self-concordant empirical loss. In: Proceedings of the Thirty Second International Conference on Machine Learning (ICML), pp. 362–370 (2015)
28. Zhuang, Y., Chin, W.S., Juan, Y.C., Lin, C.J.: Distributed Newton method for regularized logistic regression. In: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD) (2015)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.