# Bundle CDN: A Highly Parallelized Approach for Large-Scale $\ell_1$-Regularized Logistic Regression

Yatao Bian, Xiong Li, Mingqi Cao, and Yuncai Liu

Institute of Image Processing and Pattern Recognition,
Shanghai Jiao Tong University, Shanghai 200240, China
{bianyatao,lixiong,caomingqi,whomliu}@sjtu.edu.cn

**Abstract.** Parallel coordinate descent algorithms emerge with the growing demand of large-scale optimization. In general, previous algorithms are usually limited by their divergence under high degree of parallelism (DOP), or need data pre-process to avoid divergence. To better exploit parallelism, we propose a coordinate descent based parallel algorithm without needing of data pre-process, termed as Bundle Coordinate Descent Newton (BCDN), and apply it to large-scale $\ell_1$-regularized logistic regression. BCDN first randomly partitions the feature set into $Q$ non-overlapping subsets/bundles in a Gauss-Seidel manner, where each bundle contains $P$ features. For each bundle, it finds the descent directions for the $P$ features in parallel, and performs $P$-dimensional Armijo line search to obtain the stepsize. By theoretical analysis on global convergence, we show that BCDN is guaranteed to converge with a high DOP. Experimental evaluations over five public datasets show that BCDN can better exploit parallelism and outperforms state-of-the-art algorithms in speed, without losing testing accuracy.

**Keywords:** parallel optimization, coordinate descent newton, large-scale optimization, $\ell_1$-regularized logistic regression.

## 1 Introduction

High dimensional $\ell_1$-regularized models arise in a wide range of applications, such as sparse logistic regression [12] and compressed sensing [10]. Various optimization methods such as coordinate minimization [4], stochastic gradient [15] and trust region [11] have been developed to solve $\ell_1$-regularized models, among which coordinate descent newton (CDN) is proven to be promising [17].

The growing demand of scalable optimization along with the stagnant CPU speed impels people to design computers with more cores and heterogeneous computing frameworks, such as generous purpose GPU (GPGPU). To fully utilize these kinds of devices, parallel algorithms pop up like mushrooms in various areas, such as parallel annealed particle filter for motion tracking by Bian et al [1] and parallel stochastic gradient descent by Niu et al [13]. While works in [7,19] perform parallelization over samples, there are often much more features than samples in $\ell_1$-regularized problems. Bradley et al [2] proposed Shotgun CDN for $\ell_1$-regularized logistic regression by directly parallelizing the updates of features. However, Shotgun CDN is easily affected by interference among parallel updates, which limits its DOP. To get more parallelism, Scherrer et al [14] proposed to conduct feature clustering, which would introduce extra computing overhead.

To better exploit parallelism, we propose a new globally convergent algorithm, Bundle Coordinate Descent Newton (BCDN), without needing of data pre-process. In each outer iteration, BCDN randomly partitions the feature index set $\mathcal{N}$ into $Q$ subsets/bundles with size of $P$ in a Gauss-Seidel manner. In each inner iteration it first parallelly finds the descent directions for $P$ features in a bundle and second, it conducts $P$-dimensional Armijo line search to find the stepsize. A set of experiments demonstrate its remarkable properties: a highly parallelized approach with strong convergence guarantee. Experimental results with different bundle size $P$ (DOP) indicate that it could run with high DOP (large bundle size $P$). Also, its high parallelism ensures good scalability on different parallel computing frameworks (e.g. multi-core, cluster, heterogeneous computing).

The contributions of this paper are mainly threefold: (1) proposing a highly parallelized coordinate descent based algorithm, BCDN; (2) giving strong convergence guarantee by theoretical analysis; (3) applying BCDN to large-scale $\ell_1$-regularized logistic regression.

For readability, we here briefly summarize the mathematical notations as follows. $s$ and $n$ denote the number of training samples and the number of features respectively. $\mathcal{N} = \{1, 2, \cdots, n\}$ denotes the feature index set. $(\mathbf{x}_i, y_i), i = 1, \cdots, s$ denote the sample-label pairs, where $\mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, +1\}$. $\mathbf{X} \in \mathbb{R}^{s\times n}$ denotes the design matrix, whose $i^{th}$ row is $\mathbf{x}_i$. $\mathbf{w} \in \mathbb{R}^n$ is the unknown vector of model weights; $\mathbf{e}_j$ denotes the indicator vector with only the $j^{th}$ element equaling 1 and others 0. $\|\cdot\|$ and $\|\cdot\|_1$ denote the 2-norm and 1-norm, respectively.

The remainder of this paper is organized as follows. We first briefly review two related algorithms for $\ell_1$-regularized logistic regression in Section 2, then describe BCDN and its high ideal speedup in Section 3. We give the theoretical analysis for convergence guarantee of BCDN in Section 4 and present implementation and datasets details in Section 5. Experimental results will be reported in Section 6.

## 2 Algorithms for $\ell_1$-Regularized Logistic Regression

Consider the following unconstrained $\ell_1$-regularized optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^n} F_c(\mathbf{w}) \equiv c\sum_{i=1}^{s} \varphi(\mathbf{w}; \mathbf{x}_i, y_i) + \|\mathbf{w}\|_1, \tag{1}$$

where $\varphi(\mathbf{w}; \mathbf{x}_i, y_i)$ is a non-negative and convex loss function; $c > 0$ is the regularization parameter. For logistic regression, the overall training losses can be expressed as follows:

$$L(\mathbf{w}) \equiv c\sum_{i=1}^{s} \varphi(\mathbf{w}; \mathbf{x}_i, y_i) = c\sum_{i=1}^{s} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}). \tag{2}$$

A number of solvers are available for this problem. In this section, we focus on two effective solvers: CDN [17] and its parallel variant, Shotgun CDN [2].

### 2.1 Coordinate Descent Newton (CDN)

Yuan et al [17] have demonstrated that CDN is a very efficient solver for large-scale $\ell_1$-regularized logistic regression. It is a special case of coordinate gradient descent (CGD) proposed in [16]. The overall procedure of CDN is summarized in Algorithm 1.

Given the current model $\mathbf{w}$, for the selected feature $j \in \mathcal{N}$, $\mathbf{w}$ is updated along the descent direction $\mathbf{d}^j = d(\mathbf{w}; j)\mathbf{e}_j$, where,

$$d(\mathbf{w}; j) \equiv \arg\min_d \{\nabla_j L(\mathbf{w})d + \frac{1}{2}\nabla^2_{jj} L(\mathbf{w})d^2 + |w_j + d|\}. \tag{3}$$

Armijo rule is adopted based on [3] to determine the stepsize for the line search procedure. Let $\alpha = \alpha(\mathbf{w}, \mathbf{d})$ be the determined stepsize, where,

$$\alpha(\mathbf{w}, \mathbf{d}) \equiv \max_{t=0,1,2,\cdots} \{\beta^t \mid F_c(\mathbf{w} + \beta^t \mathbf{d}) - F_c(\mathbf{w}) \leq \beta^t \sigma \Delta\}, \tag{4}$$

where $0 < \beta < 1$, $0 < \sigma < 1$, $\beta^t$ denotes $\beta$ to the power of $t$, $\Delta \equiv \nabla L(\mathbf{w})^T \mathbf{d} + \|\mathbf{w}+\mathbf{d}\|_1 - \|\mathbf{w}\|_1$. This rule requires only function evaluations. According to [16], larger stepsize will be accepted if we choose $\sigma$ near 0.

**Algorithm 1.** Coordinate Descent Newton (CDN) [17]

```
1: Set $\mathbf{w}^1 = \mathbf{0} \in \mathbb{R}^n$.
2: for $k = 1, 2, 3, \cdots$ do
3:     for all $j \in \mathcal{N}$ do
4:         Obtain $\mathbf{d}^{k,j} = d(\mathbf{w}^{k,j}; j)\mathbf{e}_j$ by solving Eq. (3).
5:         Find the stepsize $\alpha^{k,j} = \alpha(\mathbf{w}^{k,j}, \mathbf{d}^{k,j})$ by solving Eq. (4).    //1-dimensional
           line search
6:         $\mathbf{w}^{k,j+1} \leftarrow \mathbf{w}^{k,j} + \alpha^{k,j}\mathbf{d}^{k,j}$.
7:     end for
8: end for
```

### 2.2 Shotgun CDN (SCDN)

Shotgun CDN (SCDN) [2] simply updates $\bar{P}$ features in parallel, where each feature update corresponds to one inner iterations in CDN, so its DOP[1] is $\bar{P}$. However, its parallel updates might increase the risk of divergence, which comes from the correlation among features. Bradley et al [2] provided a problem-specific

[1] DOP is a metric indicating how many operations can be or being simultaneously executed by a computer.

measure for SCDN's potential of parallelization: the spectral radius $\rho$ of $X^TX$. With this measure, an upper bound is given to $\bar{P}$, i.e., $\bar{P} \leq n/\rho + 1$ to achieve speedups linear in $\bar{P}$. However, $\rho$ can be very large for most large-scale datasets, e.g. $\rho = 20,228,800$ for dataset gisette with $n = 5000$, which limits the parallel ability of SCDN. Therefore, algorithms with high parallelism are desired to deal with large-scale problems. The details of SCDN can be found in Algorithm 2.

**Algorithm 2.** Shotgun CDN (SCDN) [2]

```
1: Choose the number of parallel updates $\bar{P} \geq 1$.
2: Set $\mathbf{w} = \mathbf{0} \in \mathbb{R}^n$.
3: while not converged do
4:   In parallel on $\bar{P}$ processors
5:       Choose $j \in \mathcal{N}$ uniformly at radom.
6:       Obtain $\mathbf{d}^j = d(\mathbf{w}; j)\mathbf{e}_j$ by solving Eq. (3).
7:       Find the stepsize $\alpha^j = \alpha(\mathbf{w}, \mathbf{d}^j)$ by solving Eq. (4).   //1-dimensional line search
8:       $\mathbf{w} \leftarrow \mathbf{w} + \alpha^j \mathbf{d}^j$.
9: end while
```

## 3 Bundle Coordinate Descent Newton (BCDN)

SCDN places no guarantee on its convergence when the number of features to be updated in parallel is greater than a threshold, i.e., $\bar{P} > n/\rho + 1$. This is because the 1-dimensional line search (step 7 in Algorithm 2) inside each parallel loop of SCDN cannot ensure the descent of $F_c(\mathbf{w})$ for all the $\bar{P}$ parallel feature updates. Motivated by this observation and some experimental results, we propose to perform high dimensional line search to ensure the descent of $F_c(\mathbf{w})$. The proposed method is termed as Bundle Coordinate Descent Newton (BCDN) whose overall procedure is summarized in Algorithm 3.

In each outer iteration, BCDN first randomly[2] partitions the feature index set $\mathcal{N}$ into $Q$ subsets/bundles $\mathcal{B}^1, \mathcal{B}^2, \cdots, \mathcal{B}^Q$ in a Gauss-Seidel manner,

$$\bigcup_{q=1}^{Q} \mathcal{B}^q = \mathcal{N} \quad \text{and} \quad \mathcal{B}^p \bigcap_{p \neq q} \mathcal{B}^q = \emptyset, \ \forall\ 1 < p, q < Q. \tag{5}$$

For simplicity, in practice, all bundles are set to have the same size $P$, then the number of bundles $Q = \lceil \frac{n}{P} \rceil$. Note that in the following theoretical analysis, the bundles can have different sizes.

Then in each inner iteration, BCDN first finds 1-dimensional descent directions (step 7) for features in $\mathcal{B}^q$ in parallel, then performs $P$-dimensional line search (step 10) to get the stepsize along the descent direction.

[2] The randomness is conducted by a random permutation of the feature index.

**Algorithm 3.** Bundle CDN (BCDN)

```
1: Choose the bundle size $P \in [1, n]$.
2: Set $\mathbf{w}^1 = \mathbf{0} \in \mathbb{R}^n$.
3: for $k = 1, 2, 3, \cdots$ do
4:     Randomly partition $\mathcal{N}$ to $\mathcal{B}^1, \mathcal{B}^2, \cdots, \mathcal{B}^Q$ satisfying Eq. (5).
5:     for all $\mathcal{B}^q \subseteq \mathcal{N}$ do
6:         for all $j \in \mathcal{B}^q$ in parallel do
7:             Obtain $\mathbf{d}^{k,j} = d(\mathbf{w}^{k,\mathcal{B}^q}; j)\mathbf{e}_j$ by solving Eq. (3).
8:             $\mathbf{d}^{k,\mathcal{B}^q} \leftarrow \mathbf{d}^{k,\mathcal{B}^q} + \mathbf{d}^{k,j}$.
9:         end for
10:        Find the stepsize $\alpha^{k,\mathcal{B}^q} = \alpha(\mathbf{w}^{k,\mathcal{B}^q}, \mathbf{d}^{k,\mathcal{B}^q})$ by solving Eq. (4).//$P$-dimensional
           line search
11:        $\mathbf{w}^{k,\mathcal{B}^{q+1}} \leftarrow \mathbf{w}^{k,\mathcal{B}^q} + \alpha^{k,\mathcal{B}^q}\mathbf{d}^{k,\mathcal{B}^q}$.
12:    end for
13: end for
```

Obviously, CDN is a special case of BCDN with the setting $Q = n$. That is, $\mathcal{B}^q = \{q\}, q = 1, 2, \cdots, n$.

## 3.1 High Ideal Speedup[3] of BCDN

In the following part, we will demonstrate that the ideal speedup of BCDN is the bundle size $P$, compared to CDN.

First, in the computing procedure for descent direction (step 7 in Algorithm 3), the computing of 1-dimensional descent directions for each feature is independent of each other. Therefore, the DOP is $P$ and the ideal speedup also is $P$. Then, we argue that, in BCDN, the $P$-dimensional line search (step 10 in each outer iteration in Algorithm 3) also has the ideal speedup of $P$, in comparison with CDN. In each outer iteration, BCDN runs $Q = \lceil \frac{n}{P} \rceil$ times of $P$-dimensional line search (step 10 in Algorithm 3), while CDN runs $n$ times of 1-dimensional line search (step 5 in Algorithm 1). However, the $P$-dimensional line search in BCDN costs about the same computing time as the 1-dimensional line search in CDN, which will be shown as follows.

First, each $P$-dimensional line search in BCDN will terminate roughly within the same finite number of steps, with respect to CDN. This will be proven in Theorem 1 and verified by experiments in Section 6.1. Second, the time costs of one step of line search in BCDN and CDN are equivalent. In our BCDN implementation, we maintain both $\mathbf{d}^T\mathbf{x}_i$ and $e^{\mathbf{w}^T\mathbf{x}_i}, i = 1, \cdots, s$ and follow the

[3] Here we introduce a notation "ideal speedup" to measure the speedup ratio for a parallel algorithm on an ideal computation platform. The ideal platform is assumed to have unlimited computing resources, and have no parallel schedule overhead.

**Algorithm 4.** Armijo Line Search Details

```
 1: Given $\beta, \sigma, \nabla L(\mathbf{w}), \mathbf{w}, \mathbf{d}$ and $e^{\mathbf{w}^T\mathbf{x}_i}, i = 1, \cdots, s$.
 2: $\Delta \leftarrow \nabla L(\mathbf{w})^T\mathbf{d} + \|\mathbf{w}+\mathbf{d}\|_1 - \|\mathbf{w}\|_1$.
 3: *Compute $\mathbf{d}^T\mathbf{x}_i, i = 1, \cdots, s$.
 4: for $t = 0, 1, 2, \cdots$ do
 5:    if Eq. (6) is satisfied then
 6:       $\mathbf{w} \leftarrow \mathbf{w} + \beta^t\mathbf{d}$.
 7:       $e^{\mathbf{w}^T\mathbf{x}_i} \leftarrow e^{\mathbf{w}^T\mathbf{x}_i} e^{\beta^t\mathbf{d}^T\mathbf{x}_i}, i = 1, \cdots, s$.
 8:       break
 9:    else
10:       $\Delta \leftarrow \beta\Delta$.
11:       $\mathbf{d}^T\mathbf{x}_i \leftarrow \beta\mathbf{d}^T\mathbf{x}_i, i = 1, \cdots, s$.
12:    end if
13: end for
```

implementation technique of Fan et al (see Appendix G of [5]). That is, the sufficient decrease condition in Eq. 4 is computed using the following form:

$$\begin{aligned} & f(\mathbf{w}+\beta^t\mathbf{d}) - f(\mathbf{w}) \\ & = \|\mathbf{w}+\beta^t\mathbf{d}\|_1 - \|\mathbf{w}\|_1 + c\Big(\sum_{i=1}^{s}\log\Big(\frac{e^{(\mathbf{w}+\beta^t\mathbf{d})^T\mathbf{x}_i}+1}{e^{(\mathbf{w}+\beta^t\mathbf{d})^T\mathbf{x}_i}+e^{\beta^t\mathbf{d}^T\mathbf{x}_i}}\Big) + \beta^t\sum_{i:y_i=-1}\mathbf{d}^T\mathbf{x}_i\Big) \\ & \le \sigma\beta^t(\nabla L(\mathbf{w})^T\mathbf{d} + \|\mathbf{w}+\mathbf{d}\|_1 - \|\mathbf{w}\|_1) \end{aligned} \tag{6}$$

It is worth noting that BCDN and CDN share some steps in Algorithm 4: (1) compute $\Delta$ using the pre-computed value $\nabla L(\mathbf{w})$ (step 2 in Algorithm 4); (2) in each line search step, they both check if Eq. (6) is satisfied. The only difference is the rule of computing $\mathbf{d}^T\mathbf{x}_i$ (step 3 in Algorithm 4): $\mathbf{d}^T\mathbf{x}_i = d_j x_{ij}$ in CDN because only the $j^{th}$ feature is updated, while $\mathbf{d}^T\mathbf{x}_i = \sum_{j=1}^{P} d_j x_{ij}$ in BCDN. However, $\mathbf{d}^T\mathbf{x}_i$ in BCDN could be computed in parallel with $P$ threads and a reduction-sum operation, so the time cost is equivalent for CDN and BCDN.

Summarizing the above analysis, the ideal speedup of BCDN is the bundle size $P$. It is worth noting that $P$ can be very large in practice. In our experiments, $P$ can be at least 1250 for the dataset real-sim. See Table 2 for details.

## 4 Global Convergence of BCDN

Our BCDN conducts $P$-dimensional line search for all features of a bundle $\mathcal{B}^q$ to ensure its global convergence, under high DOP. In this section, we will theoretically analyze the convergence of BCDN on two aspects: the convergence of $P$-dimensional line search and the global convergence.

**Lemma 1.** *BCDN (Algorithm 3) is a special case of CGD [16] with the specification* $H \equiv \mathrm{diag}(\nabla^2 L(\mathbf{w}))$.

*Proof.* Note that, the selection of bundle set in Eq. (5) is consistent with that used in CGD (Eq. (12) in [16]). Then, for descent direction computing for a bundle in Algorithm 3, we have,

$$\begin{aligned}
\mathbf{d}^{k,\mathcal{B}^q} &= \sum_{j\in\mathcal{B}^q} d(\mathbf{w};j)\mathbf{e}_j \\
&= \sum_{j\in\mathcal{B}^q} \arg\min_d \{\nabla_j L(\mathbf{w})^T d + \frac{1}{2}\nabla^2_{jj} L(\mathbf{w}) d^2 + |w_j + d|\}\mathbf{e}_j \qquad (7)\\
&= \arg\min_d \{\nabla L(\mathbf{w})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T H \mathbf{d} + \|\mathbf{w}+\mathbf{d}\|_1 \mid d_t = 0, \forall t \notin \mathcal{B}^q\} \qquad (8)\\
&\equiv d_H(\mathbf{w};\mathcal{B}^q) \qquad (9)
\end{aligned}$$

where Eq. (7) is derived by considering the definition of $d(\mathbf{w};j)$ in Eq. (3); Eq. (8) is obtained by applying the setting of $H \equiv \operatorname{diag}(\nabla^2 L(\mathbf{w}))$; Eq. (9) follows the definition of the descent direction by Tseng et al (Eq. (6) in [16]). Therefore the definition of direction computing is in a CGD manner.

Moreover, since BCDN conducts line Armijo search for $\mathbf{d}^{k,\mathcal{B}^q}$, it is clear that BCDN is a special case of CGD by setting $H = \operatorname{diag}(\nabla^2 L(\mathbf{w}))$. ■

By means of Lemma 1, we can use conclusions of Lemma 5 and Theorem 1(e) in [16] to prove the following theorems.

**Theorem 1 (Convergence of $P$-dimensional line search).** *For $\ell_1$-regularized logistic regression, the $P$-dimensional line search in Algorithm 3 will converge within finite steps. That is, the descent condition in Eq. (4) $F_c(\mathbf{w}+\alpha\mathbf{d}) - F_c(\mathbf{w}) \le \sigma\alpha\Delta$ is satisfied for any $\sigma \in (0,1)$ within finite steps.*

*Proof.* According to Lemma 5 of [16], to have finite steps of line search, it needs two requirements. The first requirement is,

$$\nabla^2_{jj} L(\mathbf{w}) > 0, \ \forall\, j \in \mathcal{B}^q \qquad (10)$$

and the second requirement is that there exists $E > 0$ such that,

$$\|\nabla L(\mathbf{w}_1) - \nabla L(\mathbf{w}_2)\| \le E\|\mathbf{w}_1 - \mathbf{w}_2\| \qquad (11)$$

First, we prove that the first requirement in Eq. (10) can be satisfied. Note that, we can easily obtain the closed form solution of Eq. (3):

$$d(\mathbf{w};j) = \begin{cases} -\frac{\nabla_j L(\mathbf{w})+1}{\nabla^2_{jj} L(\mathbf{w})} & \text{if } \nabla_j L(\mathbf{w}) + 1 \le \nabla^2_{jj} L(\mathbf{w}) w_j, \\ -\frac{\nabla_j L(\mathbf{w})-1}{\nabla^2_{jj} L(\mathbf{w})} & \text{if } \nabla_j L(\mathbf{w}) - 1 \ge \nabla^2_{jj} L(\mathbf{w}) w_j, \\ -w_j & \text{otherwise} \end{cases} \qquad (12)$$

where for logistic regression we have,

$$\begin{aligned}
\nabla_j L(\mathbf{w}) &= c\sum_{i=1}^{s} (\tau(y_i\mathbf{w}^T\mathbf{x}_i) - 1) y_i x_{ij} \\
\nabla^2_{jj} L(\mathbf{w}) &= c\sum_{i=1}^{s} \tau(y_i\mathbf{w}^T\mathbf{x}_i)(1-\tau(y_i\mathbf{w}^T\mathbf{x}_i)) x_{ij}^2
\end{aligned} \qquad (13)$$

where $\tau(s) \equiv \frac{1}{1+e^{-s}}$ is the derivative of the logistic loss function. Eq. (13) shows that $\nabla^2_{jj}L(\mathbf{w}) > 0$ except for $x_{ij} = 0, \forall\, i = 1, \cdots, s$. In this exception, $\nabla_j L(\mathbf{w})$ and $\nabla^2_{jj}L(\mathbf{w})$ always remain zero. There are two situations: $w_j = 0$ and $w_j \neq 0$.

- $w_j \neq 0$: according to Eq. (12), we have $d(\mathbf{w}; j) = -w_j$. Note that $d(\mathbf{w}; j) = -w_j$ also satisfies the sufficient decrease condition in Eq. (4), so $w_j$ becomes zero in the first iteration. Then, in the following iterations, $w_j$ will always satisfy the shrinking condition[4] in BCDN and will always be removed from the working set and remains zero.
- $w_j = 0$: $w_j$ will always satisfy the shrinking condition and will be removed from the working set from the first iteration to the end.

Under the above analysis, in the exception of $\nabla^2_{jj}L(\mathbf{w}) = 0$, $w_j$ will becomes zero and have no effect on the optimization procedure at least from the second iteration to the end. Therefore Eq. (10) always holds for the working set.

Second, we prove the second requirement in Eq. (11). We follow the analysis in Appendix D of [17]. For logistic regression, we have,

$$\|\nabla L(\mathbf{w}_1) - \nabla L(\mathbf{w}_2)\| \leq \|\nabla^2 L(\bar{\mathbf{w}})\| \|\mathbf{w}_1 - \mathbf{w}_2\|$$

where $\bar{\mathbf{w}} = t\mathbf{w}_1 + (1-t)\mathbf{w}_2, 0 \leq t \leq 1$. Note that, Hessian of the logistic loss can be expressed as,

$$\nabla^2 L(\mathbf{w}) = cX^T DX \tag{14}$$

where $D = \text{diag}(D_{11}, D_{22}, \cdots, D_{ss})$ with $D_{ii} = \tau(y_i \mathbf{w}^T \mathbf{x}_i)(1-\tau(y_i \mathbf{w}^T \mathbf{x}_i))$. Considering Eq. (14) and the fact that $D_{ii} < 1$, we have,

$$\|\nabla^2 L(\bar{\mathbf{w}})\| < c\|X^T\|\|X\|$$

Therefore, $E = c\|X^T\|\|X\|$ will fulfill the second requirement of Eq. (11). ■

The experimental results in Section 6.1 support the analysis in Theorem 1.

**Theorem 2 (Global Convergence of BCDN).** *Let $\{\mathbf{w}^k\}$, $\{\alpha^k\}$ be the sequences generated by Algorithm 3. If $\sup_k \alpha^k < \infty$, then every cluster point of $\{\mathbf{w}^k\}$ is a stationary point of $F_c(\mathbf{w})$.*

*Proof.* In Algorithm 4, $\alpha^k \leq 1, k = 1, 2, \cdots$, which satisfies $\sup_k \alpha^k < \infty$. To ensure the global convergence, Tseng et al made the following assumption,

$$0 < \nabla^2_{jj} L(\mathbf{w}^k) \leq \bar{\lambda},\ \forall j = 1, \cdots, n,\ k = 1, 2, \cdots \tag{15}$$

Considering Eq. (14), we have $\nabla^2_{jj}L(\mathbf{w}^k) < c\|X^T\|\|X\|$. Setting $\bar{\lambda} = c\|X^T\|\|X\|$ and following the same analysis in Theorem 1, we have $\nabla^2_{jj}L(\mathbf{w}^k) > 0$ for all features in the working set. Therefore Eq. (15) is fulfilled. According to Theorem 1(e) in [16], any cluster point of $\{\mathbf{w}^k\}$ is a stationary point of $F_c(\mathbf{w})$. ■

[4] BCDN uses the same shrinking strategy as that in CDN (Eq. (32) in [17] ).

Theorem 2 guarantees that our proposed BCDN will converge globally for any bundle size $P \in [1, n]$.

## 5 Datasets and Implementation

In this section, various aspects of the performances of CDN, SCDN and BCDN will be investigated by extensive experiments on five public datasets. For fair comparison, in these experiments, we use these methods to solve logistic regression with a bias term $b$,

$$\min_{\mathbf{w}\in\mathbb{R}^n, b} F_c(\mathbf{w}, b) \equiv c\sum_{i=1}^{s} \log(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i + b)}) + \|\mathbf{w}\|_1. \tag{16}$$

### 5.1 Datasets

The five datasets used in our experiments are summarized in Table 1[5]. news20, rcv1, a9a and real-sim are document datasets, whose instances are normalized to unit vectors. gisette is a handwriting digit problem from NIPS 2003 feature selection challenge, whose features are linearly scaled to the [-1,1] interval. The bundle size $P$ for BCDN in Algorithm 3 is set according to Table 2. Note that $P^*$ is only BCDN's conservative setting, under which BCDN can quickly converge with the most strict stopping criteria $\epsilon = 10^{-8}$ (defined in Eq. (17)). Moreover, $P$ can be larger for a common setting such as $\epsilon = 10^{-4}$ (see Section 6.4).

**Table 1.** Summary of data sets. #test is the number of test samples. The best regularization parameter $c$ is set according to Yuan et al [17]. Spa. means optimal model sparsity ($\frac{n-\|\mathbf{w}^*\|_0}{n}$), Acc. is the test accuracy for optimal model.

| Dataset | $s$ | #test | $n$ | best $c$ | Spa./% | Acc./% |
|---|---|---|---|---|---|---|
| a9a | 26,049 | 6,512 | 123 | 2 | 17.89 | 84.97 |
| real-sim | 57,848 | 14,461 | 20,958 | 4 | 83.36 | 97.16 |
| news20 | 15,997 | 3,999 | 1,355,191 | 64 | 99.80 | 95.62 |
| gisette | 6,000 | 1,000 | 5,000 | 0.25 | 90.7 | 98.10 |
| rcv1 | 541,920 | 135,479 | 47,236 | 4 | 76.77 | 97.83 |

**Table 2.** Conservative bundle size $P^*$ for each dataset

| Dataset | a9a | real-sim | news20 | gisette | rcv1 |
|---|---|---|---|---|---|
| Bundle size $P^*$ | 25 | 1,250 | 150 | 15 | 200 |

[5] All these datasets can be downloaded at
`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

### 5.2 Implementation

All the three algorithms, CDN, SCDN and BCDN, are implemented in C/C++ language. Since the shrinking procedure cannot be performed inside the parallel loop in SCDN and BCDN, to enable fair comparison, we use equivalent implementation where the shrinking is conducted outside the parallel loop in CDN. Further, we set $\sigma = 0.01$ and $\beta = 0.5$ for the line search procedure in the three algorithms. OpenMP is used as the parallel programming model. Work in parallel is distributed among a team of threads using OpenMP `parallel for` construct and the `static` scheduling of threads is used because it proves to be very efficient in our experiments.

The stopping condition similar to the outer stopping condition in [18] is used in our implementation.

$$\|\nabla^S F_c(\mathbf{w}^k)\|_1 \leq \epsilon_{end} \equiv \epsilon \cdot \frac{\min(\#pos, \#neg)}{s} \cdot \|\nabla^S F_c(\mathbf{w}^1)\|_1, \tag{17}$$

where $\epsilon$ is user-defined stopping tolerance; $\#pos$ and $\#neg$ respectively denote the numbers of positive and negative labels in the training set; $\nabla^S F_c(\mathbf{w}^k)$ is the minimum-norm sub-gradient,

$$\nabla_j^S F_c(\mathbf{w}) \equiv \begin{cases} \nabla_j L(\mathbf{w}) + 1 & \text{if } w_j > 0, \\ \nabla_j L(\mathbf{w}) - 1 & \text{if } w_j < 0, \\ \operatorname{sgn}(\nabla_j L(\mathbf{w})) \max(|\nabla_j L(\mathbf{w})| - 1, 0) & \text{otherwise} \end{cases}$$

We run CDN with an extremely small stopping criteria $\epsilon = 10^{-8}$ to get the optimal value $F_c^*$, which is used to compute the relative difference to the optimal function value (*relative error*),

$$(F_c(\mathbf{w}, b) - F_c^*)/F_c^* \tag{18}$$

Some private implementation details are listed as follows:

- CDN: we use the source code included in LIBLINEAR[6]. Shrinking procedure is modified to be consistent with the parallel algorithms BCDN and SCDN.
- SCDN: though Bradley et al [2] released the source code for SCDN, for fair comparison, we reimplement it in C language based on CDN implementation.
- BCDN: we implement BCDN carefully, including the data type and the atomic operation. For atomic operation, we use a compare-and-swap implementation using inline assembly.

## 6 Experimental Results

In this section we provide several groups of experimental results, including line search step number, scalability and timing results about relative error, testing accuracy and number of nonzeros (NNZ) in the model. To estimate the testing

[6] Version 1.7, `http://www.csie.ntu.edu.tw/~cjlin/liblinear/oldfiles/`

accuracy, for each dataset, 25% samples are used for testing and the rest samples are used for training.

All experiments are conducted on a 64-bit machine with Intel(R) Core(TM) i7 CPU (8 cores) and 12GB main memory. We set $\bar{P} = 8$ for SCDN in Algorithm 2 and use 8 threads to run the parallel updates with DOP of $\bar{P}$=8.

For BCDN, the descent direction computing (step 7 in Algorithm 3) can have the DOP of bundle size $P$ , which is several hundreds even to thousand according to Table 2, while our 8-core machine is unable to fully exhibit its parallelism potential. To justify time performances of three algorithms impartially, we need to estimate time cost for BCDN by running with at least $P$ cores. Assuming we have a machine with $P$ cores, to distribute step 7 in Algorithm 3 to $P$ cores, the extra data transfer (if needed) is little: the training data $(\mathbf{X}, \mathbf{y})$ only needs to be transferred once before all iterations, so its time cost can be omitted. Arrays with the size of $s \times sizeof(double)$ bytes containing values of $e^{\mathbf{w}^T \mathbf{x}_i}, \mathbf{d}^T \mathbf{x}_i, i = 1, \cdots, s$ need to be transferred each time, which costs very little extra transfer time. Taking into count the scheduling time, we estimate the fully parallelized computing time of descent direction computing $t_p$ by multiply the ideal parallel time cost with a reasonable factor 2,

$$t_p = (2 \cdot t_{\text{serial}})/P,$$

where $t_{\text{serial}}$ is the serial time cost of step 7 in Algorithm 3.

## 6.1 Empirical Performance of Line Search

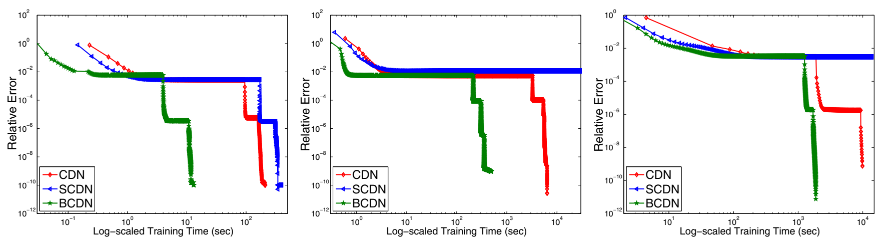
Table 3 reports the average number of line search steps per outer iteration. These statistics support the analysis in Theorem 1: for all datasets, line search of BCDN terminates in finite steps, which is far less than that of CDN and SCDN. It is also in line with the analysis in Section 3.1: BCDN conducts about $1/P$ times of line search compared to CDN, while the time cost of each line search is about the same for both. In Table 3, BCDN's number of line search steps is a little larger than $1/P$ times of that of CDN. This is because the parallel direction computing procedure in BCDN slows its convergence rate, which increases the number of line search steps. SCDN conducts more line search steps than BCDN and CDN, which indicates that its parallel strategy cannot well deal with interference among features and tends to diverge, thus needing more line search steps to fulfill the descent condition in Eq. (4).

**Table 3.** Average number of line search steps per outer iteration with $\epsilon = 10^{-4}$

| Datasets | a9a | real-sim | news20 | gisette | rcv1 |
|---|---|---|---|---|---|
| CDN | 96.2 | 3,455.3 | 3,022.3 | 472.2 | 10,272.4 |
| SCDN | 149.8 | 3,560.4 | 2,704.0 | 679.4 | 10,504.5 |
| BCDN | 6.0 | 5.7 | 62.4 | 97.6 | 57.3 |

### 6.2 Empirical Performance of Global Convergence

We verify the global convergence of all compared algorithms by setting the most strict stopping criteria $\epsilon = 10^{-8}$. Fig. 1 of the relative error (see Eq. (18)) shows that in all the cases, BCDN could converge to the final value at the fastest speed. Meanwhile, though SCDN behaves faster compared to CDN in the beginning, it cannot converge in a limited time (Fig. 1(b),(c)) or cannot converge faster than CDN (Fig. 1(a)). Table 4 with the runtime and iteration number reaches the same conclusion. The iteration number of SCDN is more than that of CDN, while the iteration number of BCDN is less than SCDN except for real-sim. This again indicates that SCDN tends to diverge with $\bar{P} = 8$ while BCDN can quickly converge under strict stopping criteria with extremely high DOP (large bundle size $P$, even to thousand).



(a) real-sim (b) news20 (c) rcv1

**Fig. 1.** Relative error under most strict stopping condition of $\epsilon = 10^{-8}$

**Table 4.** Runtime (sec) and iteration number. The number marked a "$*$" means less than that of CDN

| Methods | real-sim | | news20 | | rcv1 | |
|---|---|---|---|---|---|---|
| | time | #iter | time | #iter | time | #iter |
| CDN | 210.2 | 1,311 | 6,426.4 | 50,713 | 11,583.9 | 2,970 |
| SCDN | 399.9 | $*1,158$ | $> 54,672.9$ | $> 82,520$ | $> 13,900.6$ | $> 3,029$ |
| BCDN | $*13.2$ | 1,838 | $*489.5$ | 73,986 | $*2,298.4$ | $*2,774$ |

### 6.3 Time Performance under Common Setting

Fig. 2 plots relative error (see Eq. 18), testing accuracy and model NNZ, with a common setting of $\epsilon = 10^{-4}$. We use the conservative setting $P^*$ in Table 2 for BCDN. For all datasets, BCDN is much faster than CDN and SCDN, which highlights its higher DOP and strong convergence guarantee. Note that for gisette SCDN is even slower than CDN, which comes from its tend to diverge at a DOP of 8 ($\bar{P} = 8$).

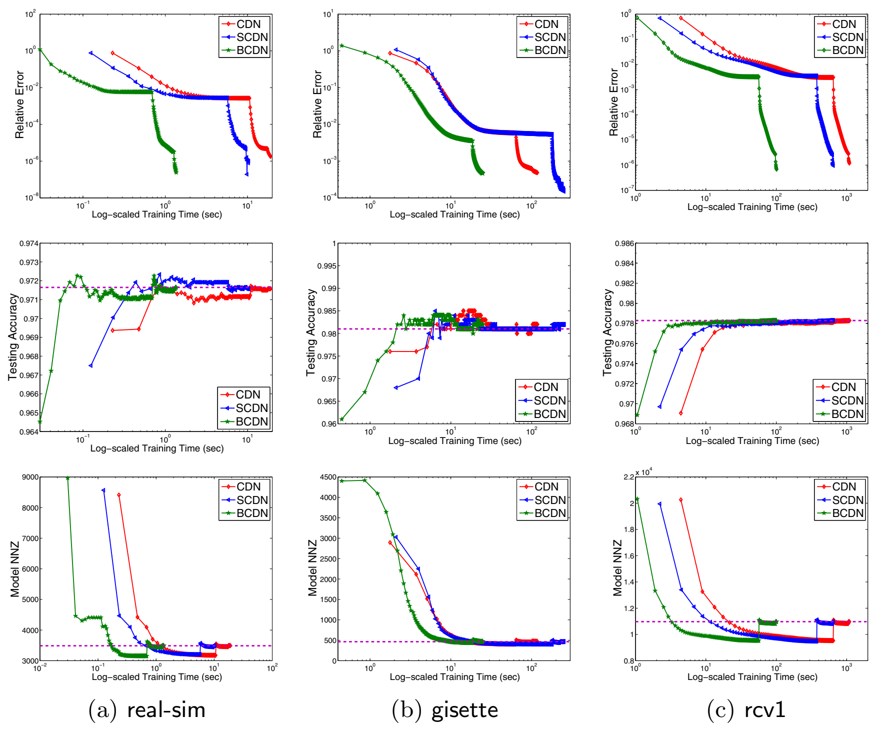


**Fig. 2.** Time performance under common setting on 3 datasets. Top, middle and bottom plot traces of relative error, testing accuracy and model NNZ over training time, respectively. The dotted horizontal line is the value obtained by running CDN with $\epsilon = 10^{-8}$.

## 6.4 Scalability of BCDN

This section evaluates the scalability of BCDN (runtime and number of outer iterations w.r.t varying bundle size $P$) with the common setting $\epsilon = 10^{-4}$. From Fig. 3 one can see that the runtime (blue lines) becomes shorter as the increase of bundle size $P$, which is in line with the analysis of BCDN in Section 3.1: BCDN has an ideal speedup of $P$. At the same time, the increase of $P$ brings about more outer iterations (see the green lines in Fig. 3), which is because more parallelism causes slower convergence rate. However, it will not introduce extra runtime because of the more parallelism with larger $P$. With the feature number of 20,958 (Table 1), real-sim could acquire an amazing high DOP of 1,300 or higher in Fig. 3 (b), which comes from the strong convergence guarantee of BCDN.

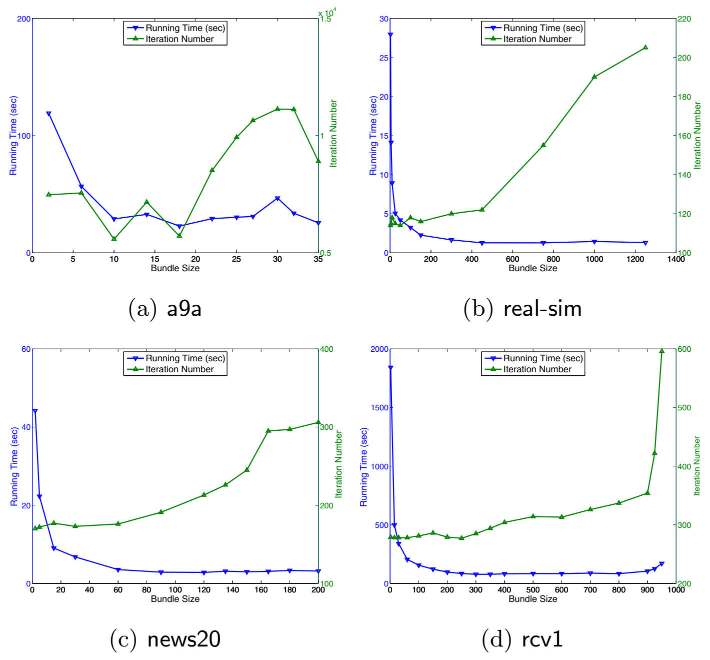


**Fig. 3.** Scalability of BCDN. Runtime (blue lines) and outer iteration number (green lines) w.r.t varying bundle size $P$.

## 7 Conclusions

This paper introduces the Bundle CDN, a highly parallelized approach with DOP of the bundle size $P$, for training high dimensional $\ell_1$-regularized logistic regression models. It has a strong convergence guarantee under theoretical analysis, which is consistent with empirical experiments. A set of experimental comparisons on 5 public large-scale datasets demonstrate that the proposed BCDN is superior among state-of-the-art $\ell_1$ solvers for speed, which comes from its high DOP to better exploit parallelism among features.

High DOP of BCDN makes it possible to develop highly parallelized algorithm on clusters with many more cores or heterogeneous computing frameworks [6] such as GPU and FPGA. BCDN can also be used to solve other $\ell_1$-regularized problems with a higher speed, such as Lasso, compressed sensing, Support Vector Machine and other related discriminative models [9,8].

**Acknowledgments.** Thanks China 973 Program 2011CB302203, STCSM Project 11JC1405200 and NSFC CB0300108 for support, and to Jiuliang Guo and Hong Wei for helpful discussions.

## References

1. Bian, Y., Zhao, X., Song, J., Liu, Y.: Parallelized annealed particle filter for real-time marker-less motion tracking via heterogeneous computing. In: ICPR, pp. 2444–2447 (2012)
2. Bradley, J.K., Kyrola, A., Bickson, D., Guestrin, C.: Parallel coordinate descent for l1-regularized loss minimization. In: ICML, pp. 321–328 (2011)
3. Burke, J.: Descent methods for composite nondifferentiable optimization problems. Math. Program. 33(3), 260–279 (1985)
4. Chang, K.W., Hsieh, C.J., Lin, C.J.: Coordinate descent method for large-scale l2-loss linear support vector machines. Journal of Machine Learning Research 9, 1369–1398 (2008)
5. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)
6. Gaster, B., Howes, L., Kaeli, D., Mistry, P., Schaa, D.: Heterogeneous Computing with OpenCL. Elsevier Science & Technology Books (2012)
7. Langford, J., Li, L., Zhang, T.: Sparse online learning via truncated gradient. Journal of Machine Learning Research 10, 777–801 (2009)
8. Li, X., Lee, T., Liu, Y.: Stochastic feature mapping for pac-bayes classification. arXiv:1204.2609 (2012)
9. Li, X., Lee, T.S., Liu, Y.: Hybrid generative-discriminative classification using posterior divergence. In: CVPR, pp. 2713–2720 (2011)
10. Li, Y., Osher, S.: Coordinate descent optimization for l1 minimization with application to compressed sensing; a greedy algorithm. Inverse Probl. Imaging 3(3), 487–503 (2009)
11. Lin, C.J., Moré, J.J.: Newton's method for large bound-constrained optimization problems. SIAM J. on Optimization 9(4), 1100–1127 (1999)
12. Ng, A.Y.: Feature selection, l1 vs. l2 regularization, and rotational invariance. In: ICML, pp. 78–85 (2004)
13. Niu, F., Recht, B., Re, C., Wright, S.J.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: NIPS, pp. 693–701 (2011)
14. Scherrer, C., Tewari, A., Halappanavar, M., Haglin, D.: Feature clustering for accelerating parallel coordinate descent. In: NIPS, pp. 28–36 (2012)
15. Shalev-Shwartz, S., Tewari, A.: Stochastic methods for $l_1$ regularized loss minimization. In: ICML, p. 117 (2009)
16. Tseng, P., Yun, S.: A coordinate gradient descent method for nonsmooth separable minimization. Math. Program. 117(1-2), 387–423 (2009)
17. Yuan, G.X., Chang, K.W., Hsieh, C.J., Lin, C.J.: A comparison of optimization methods and software for large-scale l1-regularized linear classification. Journal of Machine Learning Research 11, 3183–3234 (2010)
18. Yuan, G.X., Ho, C.H., Lin, C.J.: An improved glmnet for l1-regularized logistic regression. In: KDD, pp. 33–41 (2011)
19. Zinkevich, M., Smola, A., Langford, J.: Slow learners are fast. In: NIPS, pp. 2331–2339 (2009)