Supplementary materials for "Parallel Dual Coordinate Descent Method for Large-scale Linear Classification in Multi-core Environments"

Wei-Lin Chiang
Dept. of Computer Science
National Taiwan Univ., Taiwan
b02902056@ntu.edu.tw

Mu-Chu Lee
Dept. of Computer Science
National Taiwan Univ., Taiwan
b01902082@ntu.edu.tw

Chih-Jen Lin
Dept. of Computer Science
National Taiwan Univ., Taiwan
cjlin@csie.ntu.edu.tw

I. PROOFS

Following [1], we will apply some results proved in [4], which studies CD methods for problems in the following form:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) \\ \text{subject to} \quad & L_i \le \alpha_i \le U_i, \end{aligned} \tag{I.1}$$

where

$$f(\boldsymbol{\alpha}) \equiv g(E\boldsymbol{\alpha}) + \boldsymbol{b}^T\boldsymbol{\alpha},$$

$f(\cdot)$ and $g(\cdot)$ are proper closed functions, E is a constant matrix, and $L_i \in [-\infty, \infty)$, $U_i \in (-\infty, \infty]$ are lower/upper bounds. It has been checked in [1] that $l1$ and $l2$ loss SVM are in the form of (I.1) and satisfy additional assumptions needed in [4].

We introduce an important class of gradient-based scheme for CD's variable selection: the Gauss-Southwell rule. It plays an important role in our proof. The rule requires that at a CD step the variable i selected for update satisfies the following condition:

$$|d_i| \ge \beta \max_j |d_j|, \tag{I.2}$$

where

$$d_j = \min(\max(\alpha_j - \nabla_j f(\boldsymbol{\alpha}), 0), U) - \alpha_j,$$

and β is a fixed constant in the interval $(0, 1]$.

I.1 Proof of Theorem 1

Assume the result in Theorem 1 is wrong. Based on the use of a variable $\bar{t}$ to check the number of updates per outer iteration, we know that if the algorithm does not terminate, at lease one α_i is updated per outer iteration. Therefore, line 15 of Algorithm 4 to change $\boldsymbol{\alpha}$ is conducted infinitely many times. Let us collect all these $\boldsymbol{\alpha}$ iterates to form an infinite sequence $\{\boldsymbol{\alpha}^k\}$. If our variable selection for each CD update satisfies the Gauss-Southwell rule, then Lemma 4.2 of [4] implies that

$$\lim_{k\to\infty} \boldsymbol{\alpha}^k - P[\boldsymbol{\alpha}^k - \nabla f(\boldsymbol{\alpha}^k)] = \mathbf{0},$$

where $P[\cdot]$ is the projection operator defined as

$$P[\alpha_i] = \min(\max(\alpha_i, 0), U).$$

Then there exists $\bar{k}$ such that

$$\|\boldsymbol{\alpha}^k - P[\boldsymbol{\alpha}^k - \nabla f(\boldsymbol{\alpha}^k)]\|_\infty < \bar{\varepsilon}\min(\min_j \bar{Q}_{jj}, 1), \forall k \ge \bar{k}. \tag{I.3}$$

Note that $\min_j \bar{Q}_{jj} > 0$ because we have mentioned in Section 2.1 that instances causing $\bar{Q}_{jj} = 0$ can be easily removed before the optimization process.

We will prove that if an index i is updated at $\boldsymbol{\alpha}^k$, then

$$|\alpha_i^k - P[\alpha_i^k - \frac{\nabla_i f(\boldsymbol{\alpha}^k)}{\bar{Q}_{ii}}]| < \bar{\varepsilon}. \tag{I.4}$$

This inequality is important because it violates the condition at line 14 of Algorithm 4. We will use it to obtain the contradiction. First, if

$$0 \le \alpha_i^k - \nabla_i f(\boldsymbol{\alpha}^k) \le U, \tag{I.5}$$

then

$$\begin{aligned} &|\alpha_i^k - P[\alpha_i^k - \frac{\nabla_i f(\boldsymbol{\alpha}^k)}{\bar{Q}_{ii}}]| \\ \le &|\frac{\nabla_i f(\boldsymbol{\alpha}^k)}{\bar{Q}_{ii}}| = \frac{|\alpha_i^k - P[\alpha_i^k - \nabla_i f(\boldsymbol{\alpha}^k)]|}{\bar{Q}_{ii}} < \bar{\varepsilon}, \end{aligned} \tag{I.6}$$

where the first inequality is from the property of the projection operator, the equality is from (I.5), and the last inequality is from (I.3). Second, if

$$\alpha_i^k - \nabla_i f(\boldsymbol{\alpha}^k) < 0,$$

then with $\bar{Q}_{ii} > 0$ and (I.3)

$$\begin{aligned} &|\alpha_i^k - P[\alpha_i^k - \frac{\nabla_i f(\boldsymbol{\alpha}^k)}{\bar{Q}_{ii}}]| \\ \le &|\alpha_i^k| = |\alpha_i^k - P[\alpha_i^k - \nabla_i f(\boldsymbol{\alpha}^k)]| < \bar{\varepsilon}. \end{aligned} \tag{I.7}$$

The situation for

$$\alpha_i^k - \nabla_i f(\boldsymbol{\alpha}^k) > U$$

is similar. Therefore we have (I.4). However, (I.4) violates our condition to update α_i^k. Thus our assumption is wrong and the algorithm should terminate after a finite number of steps.

It remains to prove that our variable selection follows the Gauss-Southwell rule. Because $\boldsymbol{\alpha}$ is in the following compact set,[1]

$$\{\boldsymbol{\alpha} \mid f(\boldsymbol{\alpha}) \le f(\text{initial } \boldsymbol{\alpha}), \boldsymbol{\alpha} \text{ is feasible}\}, \tag{I.8}$$

[1] See a proof in, for example [1, Section 7.1].

there exists a constant S such that for all $\boldsymbol{\alpha}$ in the set defined in (I.8),

$$\max_j |\alpha_j - P[\alpha_j - \nabla_j f(\boldsymbol{\alpha})]| \le S.$$

Let

$$\beta = \frac{\bar{\varepsilon}\min(1, \min_j \bar{Q}_{jj})}{S}. \tag{I.9}$$

Then we have that at any iteration k, the selected index i satisfies

$$\begin{aligned}
&|\alpha_i^k - P[\alpha_i^k - \nabla_i f(\boldsymbol{\alpha}^k)]| \\
\ge &\min(1, \min_j \bar{Q}_{jj})(\alpha_i^k - P[\alpha_i^k - \frac{\nabla_i f(\boldsymbol{\alpha}^k)}{\bar{Q}_{ii}}]) \\
\ge &\min(1, \min_j \bar{Q}_{jj})\bar{\varepsilon} \\
= &\beta S \\
\ge &\beta \max_j |\alpha_j^k - P[\alpha_j^k - \nabla_j f(\boldsymbol{\alpha}^k)]|.
\end{aligned}$$

The first inequality comes from properties of the projection operator; see how we derive the first inequality in (I.6) and (I.7). The second inequality is from how we decided if an element should be updated or not, while the last is from (I.9). Therefore, our selection follows the Gauss-Southwell rule.

I.2 Proof of Theorem 2

Assume the result is wrong. Because we have shown in Section I.1 that $\{\alpha^{\varepsilon_k,\bar{\varepsilon}_k}\}$ is in a compact set, there is a convergent sub-sequence $\{\boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k}\}, k \in K$ such that

$$\lim_{k\in K, k\to\infty} \boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k} = \bar{\boldsymbol{\alpha}} \tag{I.10}$$

and

$$\lim_{k\in K, k\to\infty} \boldsymbol{w}^{\varepsilon_k,\bar{\varepsilon}_k} = \bar{\boldsymbol{w}} = \sum\nolimits_{j=1}^{l} \bar{\alpha}_j y_j \boldsymbol{x}_j \ne \boldsymbol{w}^*.$$

Before Algorithm 4 stops, at the last iteration, we have the following intermediate vectors.

$$\boldsymbol{\alpha}^{k,1}, \ldots, \boldsymbol{\alpha}^{k,T}, \boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k},$$

where $\boldsymbol{\alpha}^{k,t}$ corresponds to the $\boldsymbol{\alpha}$ vector before the set $\bar{B}_t$ is handled. We will prove that

$$\begin{aligned}
\lim_{k\in K, k\to\infty} \boldsymbol{\alpha}^{k,1} = \cdots &= \lim_{k\in K, k\to\infty} \boldsymbol{\alpha}^{k,T} \\
&= \lim_{k\in K, k\to\infty} \boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k} = \bar{\boldsymbol{\alpha}}.
\end{aligned} \tag{I.11}$$

Consider $\boldsymbol{\alpha}^{k,t}$ and $\boldsymbol{\alpha}^{k,t+1}$. Between these two vectors elements in $\bar{B}_t$ may be updated. We can further consider the following iterates

$$\boldsymbol{\alpha}^{k,t,1}, \ldots, \boldsymbol{\alpha}^{k,t,|\bar{B}_t|}. \tag{I.12}$$

Because only elements in the selected subset $B \subset \bar{B}_t$ are actually considered for update, many adjacent ones in (I.12) are the same. Regardless of whether $\boldsymbol{\alpha}^{k,t,s} = \boldsymbol{\alpha}^{k,t,s+1}$ or not, from Lemma 2 in Section 7.4 of [1],

$$\begin{aligned}
f(\boldsymbol{\alpha}^{k,t,s}) - f(\boldsymbol{\alpha}^{k,t,s+1}) &\ge \frac{1}{2}\bar{Q}_{ii}\|\boldsymbol{\alpha}^{k,t,s} - \boldsymbol{\alpha}^{k,t,s+1}\| \\
&\ge \frac{1}{2}\min_j \bar{Q}_{jj}\|\boldsymbol{\alpha}^{k,t,s} - \boldsymbol{\alpha}^{k,t,s+1}\|,
\end{aligned}$$

where $\boldsymbol{x}_i$ is assumed to be the instance considered at $\boldsymbol{\alpha}^{k,t,s}$. Because our setting ensures that the function value is monotonically decreasing and $f(\boldsymbol{\alpha})$ is lower-bounded, we have that $f(\boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k})$ as well as $f(\boldsymbol{\alpha}^{k,t,s}), \forall s$ globally converge. Therefore,

$$\begin{aligned}
&\lim_{k\in K, k\to\infty} f(\boldsymbol{\alpha}^{k,t,s}) - f(\boldsymbol{\alpha}^{k,t,s+1}) = 0 \\
&= \lim_{k\in K, k\to\infty} \frac{1}{2}\min_j \bar{Q}_{jj}\|\boldsymbol{\alpha}^{k,t,s} - \boldsymbol{\alpha}^{k,t,s+1}\|.
\end{aligned}$$

Note that we have explained in Section 2.1 that $\min_j \bar{Q}_{jj} > 0$. Then the limits of all vectors in (I.12) when $k \in K, k \to \infty$ are all the same. Therefore,

$$\lim_{k\in K, k\to\infty} \boldsymbol{\alpha}^{k,t} = \lim_{k\in K, k\to\infty} \boldsymbol{\alpha}^{k,t+1}.$$

By similar arguments, we have (I.11).

When Algorithm 4 stops, we see that either

$$|\nabla_i^P f(\boldsymbol{\alpha}^{k,t})| \le \varepsilon_k, \forall t = 1, \ldots, T, \forall i \in \bar{B}_t \tag{I.13}$$

or

$$\begin{aligned}
&\boldsymbol{\alpha}^{k,1} = \cdots = \boldsymbol{\alpha}^{k,T} = \boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k} \text{ and} \\
&\begin{cases} |\nabla_i^P f(\boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k})| \le \delta\varepsilon_k, \text{ or} \\ \alpha_i^{\varepsilon_k,\bar{\varepsilon}_k} - P\left[\alpha_i^{\varepsilon_k,\bar{\varepsilon}_k} - \nabla_i f(\boldsymbol{\alpha}^{\varepsilon_k,\bar{\varepsilon}_k})\right] \le \bar{\varepsilon}_k, \end{cases} \quad \forall i = 1, \ldots, l.
\end{aligned} \tag{I.14}$$

The first case corresponds to the situation when $M \le \varepsilon$ holds at line 18, while the second situation means that $\bar{t} = 0$ occurs (either B is empty in selecting elements from $\bar{B}$ at line 10 or $|d| < \bar{\varepsilon}$ at line 14.)

Because $\bar{\boldsymbol{\alpha}}$ is not optimal, from the optimality condition there exists an index i such that

$$\begin{aligned}
&\nabla_i f(\bar{\boldsymbol{\alpha}}) < 0 \text{ if } \bar{\alpha}_i = 0, \text{or} \\
&\nabla_i f(\bar{\boldsymbol{\alpha}}) > 0 \text{ if } \bar{\alpha}_i = U, \text{or} \\
&\nabla_i f(\bar{\boldsymbol{\alpha}}) \ne 0 \text{ if } 0 < \bar{\alpha}_i < U.
\end{aligned} \tag{I.15}$$

From (16), the continuity of $\nabla f(\boldsymbol{\alpha})$, and (I.11), there exists $\bar{k}$ such that for all $k \in K, k \ge \bar{k}, \forall t = 1, \ldots, T$

Case 1 of (I.15):

$$0 \le \alpha_i^{k,t} + \varepsilon_k < U, \nabla_i f(\boldsymbol{\alpha}^{k,t}) < -\varepsilon_k.$$

Case 2 of (I.15):

$$0 < \alpha_i^{k,t} - \varepsilon_k \le U, \nabla_i f(\boldsymbol{\alpha}^{k,t}) > \varepsilon_k.$$

Case 3 of (I.15):

$$\varepsilon_k < \alpha_i^{k,t} < U - \varepsilon_k, |\nabla_i f(\boldsymbol{\alpha}^{k,t})| > \varepsilon_k.$$

From these three cases and our setting of $\varepsilon_k > \bar{\varepsilon}_k$, we have

$$|\nabla_i^P f(\boldsymbol{\alpha}^{k,t})| > \varepsilon_k, \forall t = 1, \ldots, T,$$

and

$$\alpha_i^{k,t} - P\left[\alpha_i^{k,t} - \nabla_i f(\boldsymbol{\alpha}^{k,t})\right] > \varepsilon_k > \bar{\varepsilon}_k, \forall t = 1, \ldots, T.$$

This clearly violates (I.13) and (I.14). Therefore, our assumption is wrong, and hence $\{\boldsymbol{w}^{\varepsilon_k,\bar{\varepsilon}_k}\}$ converges to the optimal $\boldsymbol{w}^*$.

II. ADDITIONAL ANALYSIS OF ALGORITHM 4

II.1 Scheduling of the Parallel For Loop

A parallel **for** loop must assign tasks to different threads. For example, OpenMP may statically dispatch tasks to threads or dynamically assign tasks according to the load of each thread. The setting, often referred to as the scheduling of the loop, may affect the computational speed; see, for example, a study in [3, Supplement]. We applied different OpenMP scheduling schemes for the operation at line 8 of Algorithm 4. Results show that the running time is about the same. The explanation is that the **for** loop being parallelized is a light task: Because $\bar{B}$ is relatively small (no more than a few thousands), it is difficult to improve the utilization of cores by improving the load balance.

III. DETAILED RESULTS OF MINI-BATCH CD

In Figure III, we compare mini-batch CD using atomic and reduce operations for updating $\boldsymbol{w}$ (see Section 2.2.1) with LIBLINEAR. The parameter β_b in Algorithm 2 for mini-batch CD is set to be

$$\beta_b = 1 + \frac{(b-1)(l\sigma^2 - 1)}{l-1}, \tag{III.1}$$

where b is the batch size, l is the number of instances, and σ^2 is the spectral norm of the normalized data matrix

$$\bar{X} = [\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l],$$

in which each $\bar{\boldsymbol{x}}_i = \boldsymbol{x}_i / \|\boldsymbol{x}_i\|$.[2] We implemented the mini-batch CD without the shrinking technique because the expected convergence proof in [5] is based on the randomness of all instances. For a fair comparison, LIBLINEAR without shrinking is used.

An important parameter to be decided is the size of B. In (III.1), we can observe that $|B| = b$ is strongly related to β_b, which is an important coefficient of the sub-problem in Algorithm 2. If $|B|$ is too large, then the CD update in each iteration becomes more conservative, leading to a slow convergence. In contrast, if $|B|$ is too small, then the overhead in parallelizing CD updates becomes significant, a situation that may lead to a worse scalability.

We consider three sizes of $|B|$: 16, 64 and 256. The results show that for the dense data covtype, a small $|B|$ leads to no scalability for both implementations. Further, the atomic operations significantly slow down the program in all sizes of B. For the sparse data rcv1, the opposite result is observed: the one implementing the reduce operation is worse because each dense array $\hat{\boldsymbol{u}}^p$ defined in (9) handles only several sparse instances. For the comparison with the single-core LIBLINEAR without the shrinking technique, clearly the mini-batch method is slower. This result leads to our decision in Section 5.3 for not including mini-bach CD in the main comparison. The experimental result also confirms our assessment in Section 2.2.1, in which we point out the difficulty to update some $\boldsymbol{w}$ components together in a multi-core environment.

IV. DETAILS OF THE SHRINKING IMPLEMENTATION

[2]The computational time presented in Figure III does not include the calculation of σ^2, which is quite time-consuming.

In Algorithm I we give details of implementing Algorithm 4 with the shrinking technique. We directly apply the setting in [1], so Algorithm I is basically the combination of Algorithm 4 in this work and Algorithm 3 in [1].

We mentioned in Section 3.1 that LIBLINEAR actually uses

$$\max_i \nabla f(\boldsymbol{\alpha}^{k,i}) - \min_i \nabla f(\boldsymbol{\alpha}^{k,i}) < \varepsilon$$

as the stopping condition. We follow the same setting so at line 32 the condition becomes $M - m \le \varepsilon$. In the future we hope to change the dual CD in LIBLINEAR and the parallel extension to use

$$\max_i |\nabla_i^P f(\boldsymbol{\alpha}^{k,i})| < \varepsilon.$$

V. RESULTS WITHOUT APPLYING THE SHRINKING TECHNIQUE

Under the same setting in Section 5.3, we compare asynchronous CD, Algorithm 4, and LIBLINEAR without applying the shrinking technique.

However, some issues occur in asynchronous CD without applying the shrinking technique because we found that the implementation in [2] has very slow final convergence. In [1, Section 3.1], it is noticed that a random permutation of indices in the beginning of each outer iterations leads to much faster convergence than the setting of using a fixed order of CD updates. However, the experimental code in [2] did not conduct the same procedure of random shuffling as LIBLINEAR. Their setting is to split all instances into P blocks, where P is the number of threads, and all threads parallely permute indices within their associated blocks. Therefore, elements in each block remain fixed throughout iterations. To analyze the influence of the randomness, we modified the experimental code in [2] to have the same global index permutation as in LIBLINEAR. A comparison between the implementation in [2] and the new setting is in Figure IV. We can observe that for almost all data, asynchronous CD with a weaker randomness converges very slowly in the later stage.

As a comparison, we check the situation when the shrinking technique is used. Results are shown in Figure V. The difference between the two settings is less significant than that in Figure IV. The explanation is that the randomness is improved by the shrinking technique. In LIBLINEAR as well as the implementation in [2], they move shrunken elements to the end of the index list. Therefore, when the shrinking technique is applied, the order of instances may be changed at each iteration, leading to a better randomness.

Based on the above analysis, for the comparison with Algorithm 4 and LIBLINEAR without the shrinking technique we consider the new asynchronous CD implementation that permutes all indices in the beginning of each outer iteration. Results are in Figures VI and VII for $l1$ and $l2$ losses, respectively. We observe that the scalability of Algorithm 4 is better than that in Figure 1. The explanation is that without the shrinking technique, many elements (e.g., those that will eventually be bounded) have $\nabla_i^P f(\boldsymbol{\alpha}) = 0$. Then the size of B is relatively smaller in comparison with the size of $\bar{B}$. Hence a better scalability is obtained. For asynchronous CD, the convergence is improved in url-combined. Our guess is that because many indices have $\nabla_i^P f(\boldsymbol{\alpha}) = 0$, after the gradient value is calculated, the thread does not need to update $\boldsymbol{w}$. From the less frequent update of $\boldsymbol{w}$, the

Algorithm I A parallel dual CD method in practice

Specify $\boldsymbol{\alpha}$ and calculate $\boldsymbol{w} = \sum_j y_j \alpha_j \boldsymbol{x}_j$
Specify $\delta, \varepsilon, \bar{\varepsilon}, \text{initB}, \text{maxB}$
Let $\bar{M} \leftarrow \infty, \bar{m} \leftarrow -\infty, A \leftarrow \{1, \ldots, l\}$
$\text{nowB} \leftarrow \text{initB}$
while true **do**
  Let $M \leftarrow -\infty, m \leftarrow \infty, \bar{A} \leftarrow A, \bar{t} \leftarrow 0$
  while $\bar{A} \neq \emptyset$ **do**
    Choose $\bar{B} \subset \bar{A}$ with $|\bar{B}| = \min(\text{nowB}, |\bar{A}|)$
    $\bar{A} \leftarrow \bar{A} \setminus \bar{B}$
    Calculate $\nabla f_{\bar{B}}(\boldsymbol{\alpha})$ in parallel
    $B \leftarrow \emptyset$
    for $i \in \bar{B}$ **do**
      $PG \leftarrow 0$; $G \leftarrow \nabla_i f(\boldsymbol{\alpha})$
      if ($\alpha_i < C$ and $G < 0$) or ($\alpha_i > 0$ and $G > 0$) **then**
        $PG \leftarrow G$
      else if ($\alpha_i = 0$ and $G > \bar{M}$) or ($\alpha_i = C$ and $G < \bar{m}$) **then**
        $A \leftarrow A \setminus \{i\}$
      $M \leftarrow \max(M, PG), m \leftarrow \min(m, PG)$
      if $|PG| \geq \delta \varepsilon_1$ **then**
        $B \leftarrow B \cup \{i\}$
    if $|B| = 0$ **then**
      $\text{nowB} \leftarrow \min(1.5 \times \text{nowB}, \text{maxB})$
    else if $|B| \geq \text{initB}$ **then**
      $\text{nowB} \leftarrow \text{nowB}/2$
    for $i \in B$ **do**
      $G \leftarrow y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1 + D_{ii} \alpha_i$
      $d \leftarrow \min(\max(\alpha_i - G/\bar{Q}_{ii}, 0), U) - \alpha_i$
      if $|d| \geq \bar{\varepsilon}$ **then**
        $\alpha_i \leftarrow \alpha_i + d$
        $\boldsymbol{w} \leftarrow \boldsymbol{w} + d y_i \boldsymbol{x}_i$
        $\bar{t} \leftarrow \bar{t} + 1$
  if $M - m \leq \varepsilon_1$ or $\bar{t} = 0$ **then**
    if $A = \{1, \ldots, l\}$ and $\varepsilon_1 \leq \varepsilon$ **then**
      break
    else
      $A \leftarrow \{1, \ldots, l\}, \bar{M} \leftarrow \infty, \bar{m} \leftarrow -\infty$
      $\varepsilon_1 \leftarrow \max(0.1\varepsilon_1, \varepsilon)$
  if $M \leq 0$ **then**
    $\bar{M} \leftarrow \infty$
  else
    $\bar{M} \leftarrow M$
  if $m \geq 0$ **then**
    $\bar{m} \leftarrow -\infty$
  else
    $\bar{m} \leftarrow m$

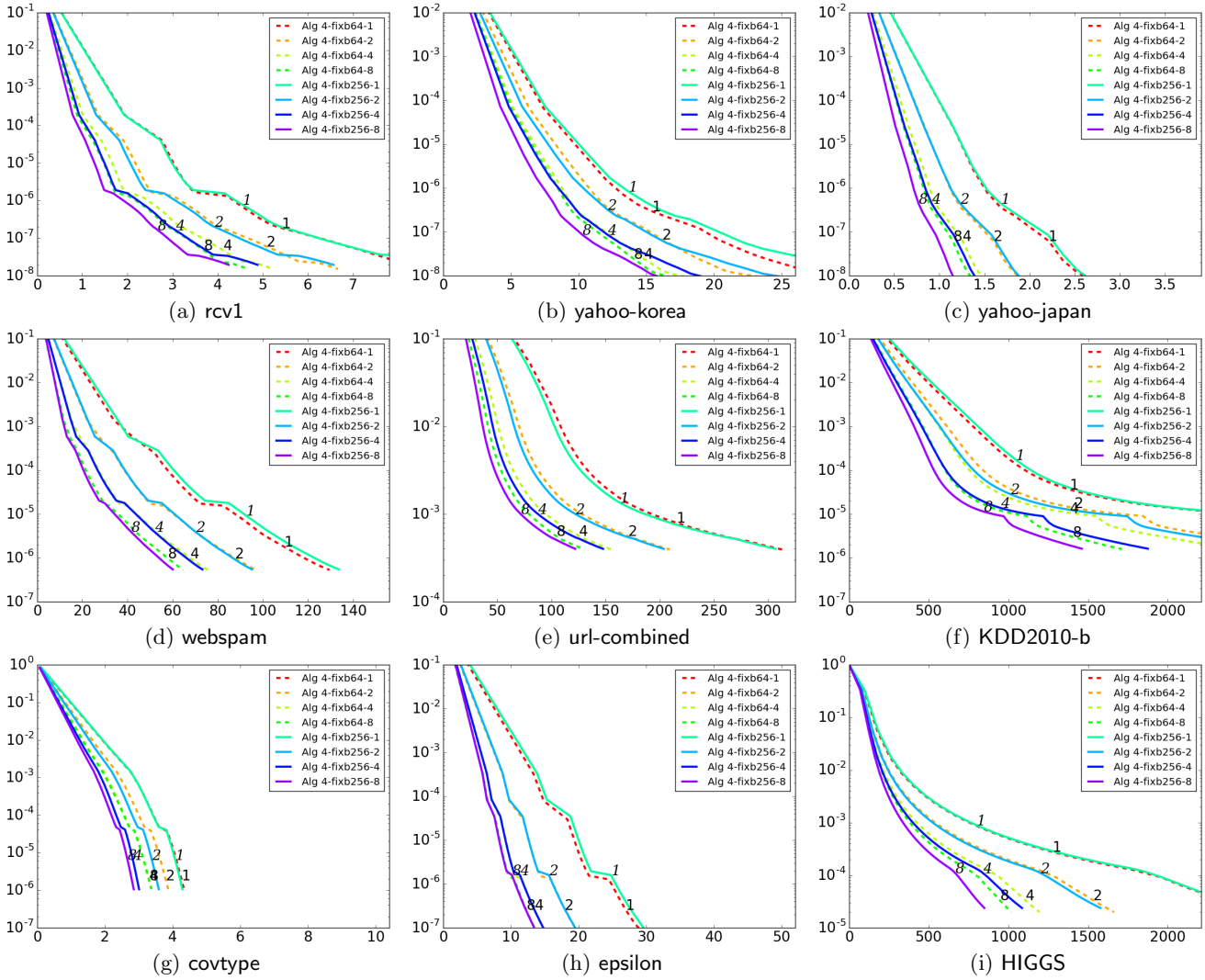


Figure I: A comparison of Algorithm 4 with $|\bar{B}| = 64$ and 256. In the legend, "fix" means that the adaptive rule in Section 4.2 is not applied to update $|\bar{B}|$. All other settings are the same as Figure 1. The $l1$ loss is considered.

lag τ discussed in Section 2.2.2 is relatively small and may become smaller and hence the desired conditions for the convergence are easier satisfied. Unfortunately, asynchronous CD still fails for covtype. In summary, when shrinking is not applied, Algorithm 4 is competitive with asynchronous CD and is still more robust.

VI. RESULTS OF USING DIFFERENT C VALUES

In Section 5 we present results of $C = 1$. We wonder if similar observations can still be made under other C values. While users may experiment with different values, an important C value is the one that achieves the best validation accuracy. Therefore, we conduct five-fold cross validation on $C \in \{2^{-10}, \ldots, 2^{10}\}$. Then the best C is used for comparing the training time. Results for $l1$ and $l2$ losses are respectively represented in Figures VIII and IX. All other settings are the same as those in Section 5. Results are generally similar to those in Figures 1 and 2 because in many cases the selected C is not very different from $C = 1$. However, we roughly see that the problems become more difficult when C is large (e.g., webspam, covtype, epsilon in Figure VIII and webspam in Figure IX). It is known that the convergence of dual CD is slower for such cases, but how scalability is affected is an issue worth investigating.

VII. THE STOPPING CONDITION OF ALGORITHM I

To show the relationship between the training time and the closeness to the optimal object value, we apply non-stop settings for all approaches in Figures 1 and 2. For Algorithm I (indicated as Algorithm 4 in the paper), we set $\varepsilon = 0$ and initial $\varepsilon_1 = 0.1$. Therefore, whenever $M - m \le \varepsilon_1$ is satisfied, ε_1 is reduced by a factor of 10. For asynchronous CD and LIBLINEAR, we initially set the stopping tolerance ε to be 0.1. When $M - m \le \varepsilon$ is satisfied, we keep reducing ε by a factor of 10.

In the practical use, we can not apply a non-stop setting.

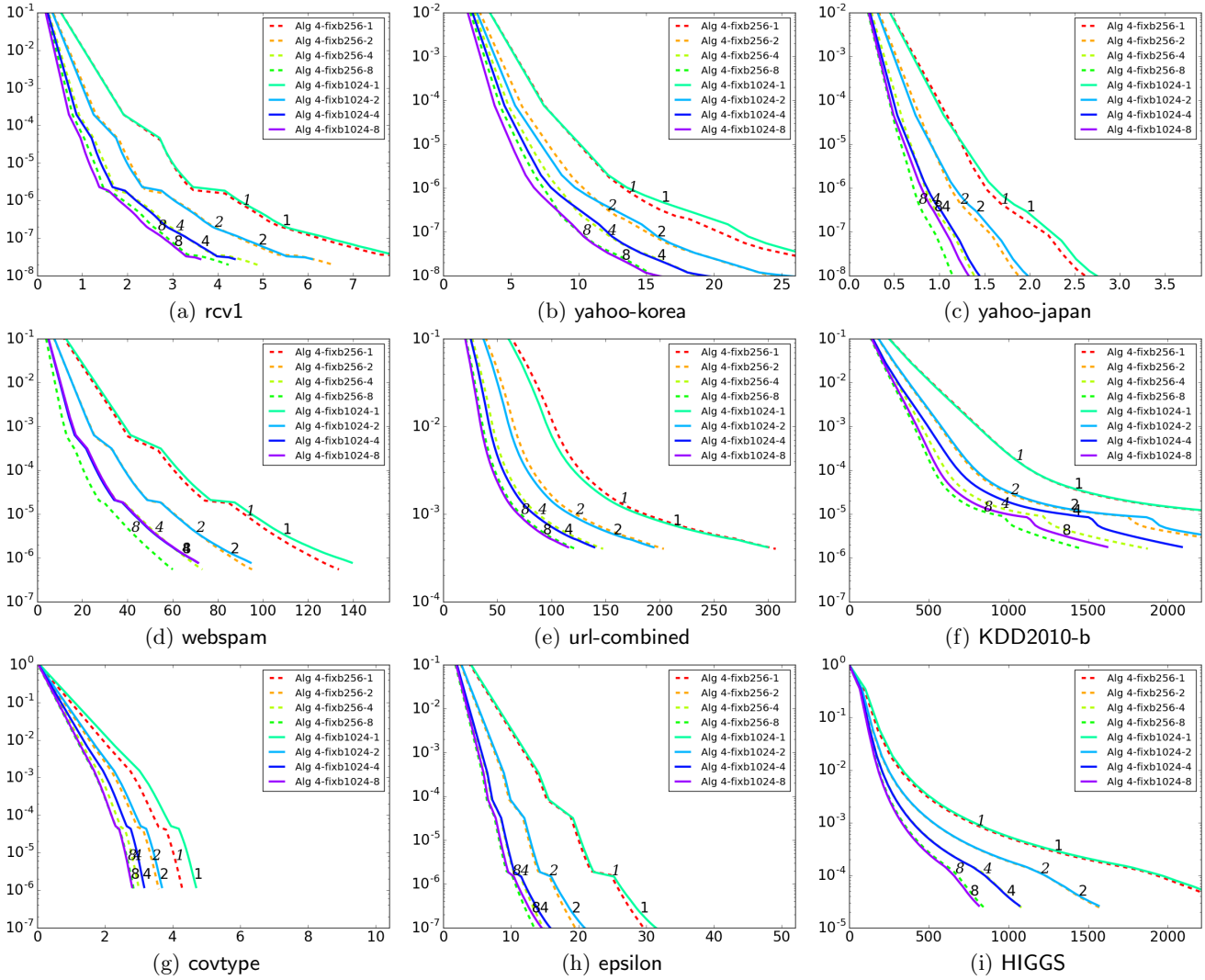


Figure II: A comparison of Algorithm 4 with $|\bar{B}| = 256$ and 1024. In the legend, "fix" means that the adaptive rule in Section 4.2 is not applied to update $|\bar{B}|$. All other settings are the same as Figure 1. The $l1$ loss is considered.

For Algorithm I, this means that a stopping condition under a given $\varepsilon > 0$ is used. It is important to check if the algorithm performs well. In this section, we give some detailed analysis and study the behavior under different stopping conditions. Finally, we make an improved version of Algorithm I regarding the issues when stopping conditions are considered.

In Section 4.3, we introduced a variable ε_1 to prevent Algorithm I from being ε-dependent. However, ε is still a lower bound of ε_1 (see line 37 in Algorithm I). Therefore, the behavior of Algorithm I is still affected by different ε. To see the effect of ε, we run Algorithm I under $\varepsilon = 0.1$ and 0.01. Results are shown in Figures XI and XII respectively. Clearly, we can observe that Algorithm I converges slower in some periods, particularly in the final stage (e.g., some almost horizontal segments in the end of the curves of "Alg I-1" for problems rcv1, webspam, and covtype). There are two possible reasons. First, resetting the active set A is a time-consuming process because all gradient elements including those which should not be checked are calculated in the next iteration. In Algorithm I, we reset it whenever ε_1 is reduced (see line 36); this may be too frequent. Next, we describe another reason. Because of the setting

$$\varepsilon_1 \leftarrow \max(0.1\varepsilon_1, \varepsilon),$$

in the final stage of the algorithm, we may have

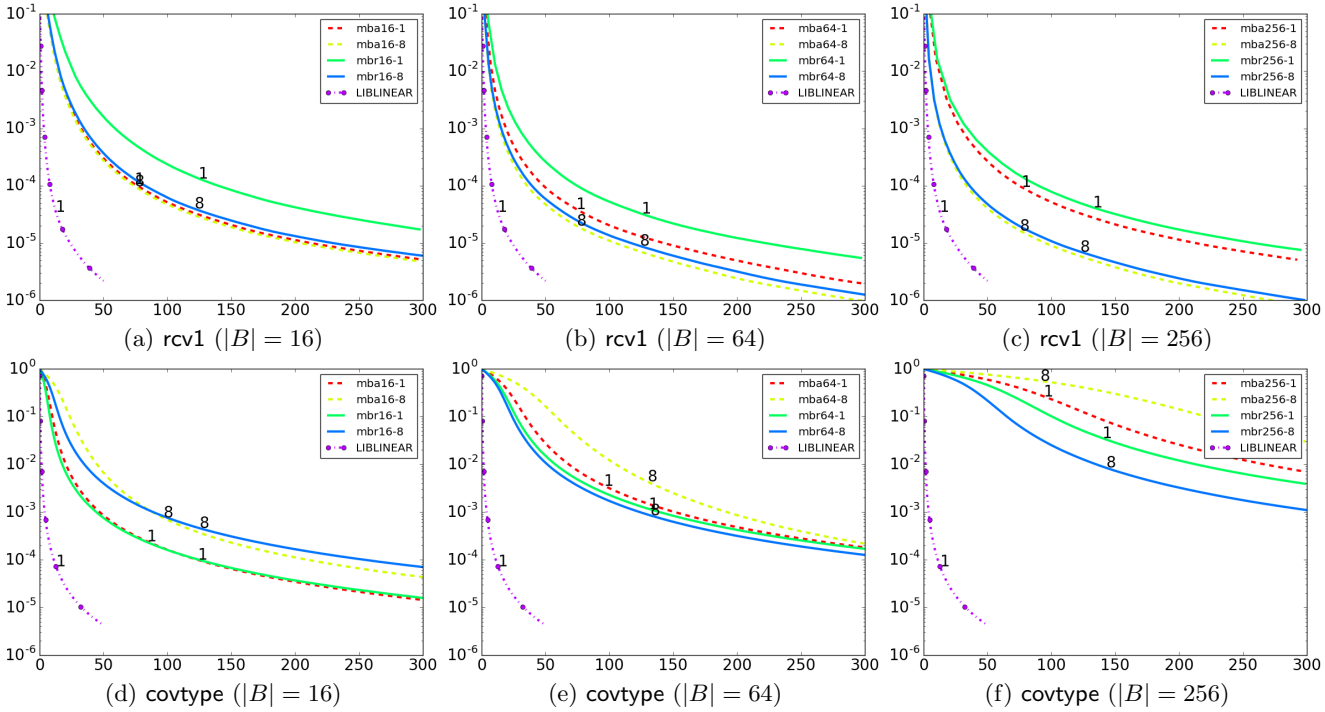
$$\varepsilon_1 = \varepsilon.$$

Then with shrinking, the condition

$$M - m \le \varepsilon_1$$

may be quickly satisfied after only very few $\boldsymbol{\alpha}$ elements are updated. This process may repeat several times until both $M - m \le \varepsilon$ and $A = \{1, \ldots, l\}$ are true.

We revise the **if** statement in lines 40-45 of Algorithm I to prevent the frequent reset of the active set A, while still make the decrease of ε_1 possible. To begin, we make the lower bound of ε_1 be smaller than ε by using 0.01ε. Second,



(a) rcv1 ($|B| = 16$) (b) rcv1 ($|B| = 64$) (c) rcv1 ($|B| = 256$)

(d) covtype ($|B| = 16$) (e) covtype ($|B| = 64$) (f) covtype ($|B| = 256$)

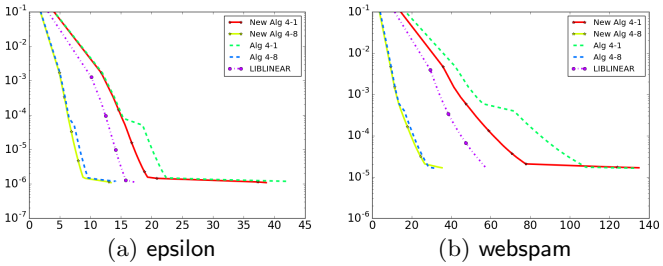
Figure III: A comparison between single-core LIBLINEAR and multi-core mini-batch CD. The mini-batch CD implementations by using atomic and reduce operations are respectively denoted as "mba" and "mbr." We present running time in seconds (x-axis) versus the relative difference to the optimal objective value (y-axis, log-scaled). All methods are implemented without the shrinking technique. We use 1 and 8 cores for mini-batch CD.

the reset of the active set A occurs only if

$$M - m \leq \varepsilon$$

holds rather than when ε_1 is decreased. Details are in the following statements.

1: Specify $\varepsilon_1 = 0.1$
2: Specify $\varepsilon_{\min} = \min(0.01\varepsilon, \varepsilon_1)$
3: // skipped
4: **if** $M - m \leq \varepsilon_1$ or $\bar{t} = 0$ **then**
5: $\quad \varepsilon_1 \leftarrow \max(0.1\varepsilon_1, \varepsilon_{\min})$
6: **if** $M - m \leq \varepsilon$ **then**
7: $\quad$ **if** $A = \{1, \ldots, l\}$ and $\varepsilon_1 \leq \varepsilon$ **then**
8: $\quad\quad$ break
9: $\quad$ **else**
10: $\quad\quad$ $A \leftarrow \{1, \ldots, l\}, \bar{M} \leftarrow \infty, \bar{m} \leftarrow -\infty$



(a) epsilon (b) webspam

Figure X: A comparison between Algorithm I and the new setting. We set $\varepsilon = 0.01$ and the $l1$ loss is used.

In Figure X, we show the results of applying new settings. We can observe that the less frequent reactivation of the set A does not affect the convergence speed. However, the slow convergence in the final stage may still be observed. The reason is that when

$$M - m \leq \varepsilon$$

holds, the situation is similar to when $\varepsilon_1 = \varepsilon$ occurs in the previous setting. Our earlier discussion has explained that slow convergence may happen. To improve the final convergence, we further reduce the frequency of reactivating the set A by modifying line 6 to be

$$M - m \leq \rho\varepsilon, \tag{VII.1}$$

where $\rho < 1$ but is close to 1. The reason behind this setting is that

$$M - m \leq \varepsilon \tag{VII.2}$$

is a condition applied on a smaller problem of only variables in A. It is easier to hold than a condition on all variables. When (VII.2) is satisfied, we may not be that close to the optimal solution yet and hence the reactivation of the set A is not necessary. On the other hand, if the shrinking procedure does not remove any elements, our new setting will lead to longer training time. Therefore, the parameter ρ should be only slightly smaller than 1. In Figures XI and XII, we present the result of using $\rho = 0.9$. Clearly, we can see that in the final stage, the training time is significantly improved for many data sets. Another observation is that when the number of cores is increased from one to eight, the improvement becomes less dramatic. The reason is that the slow

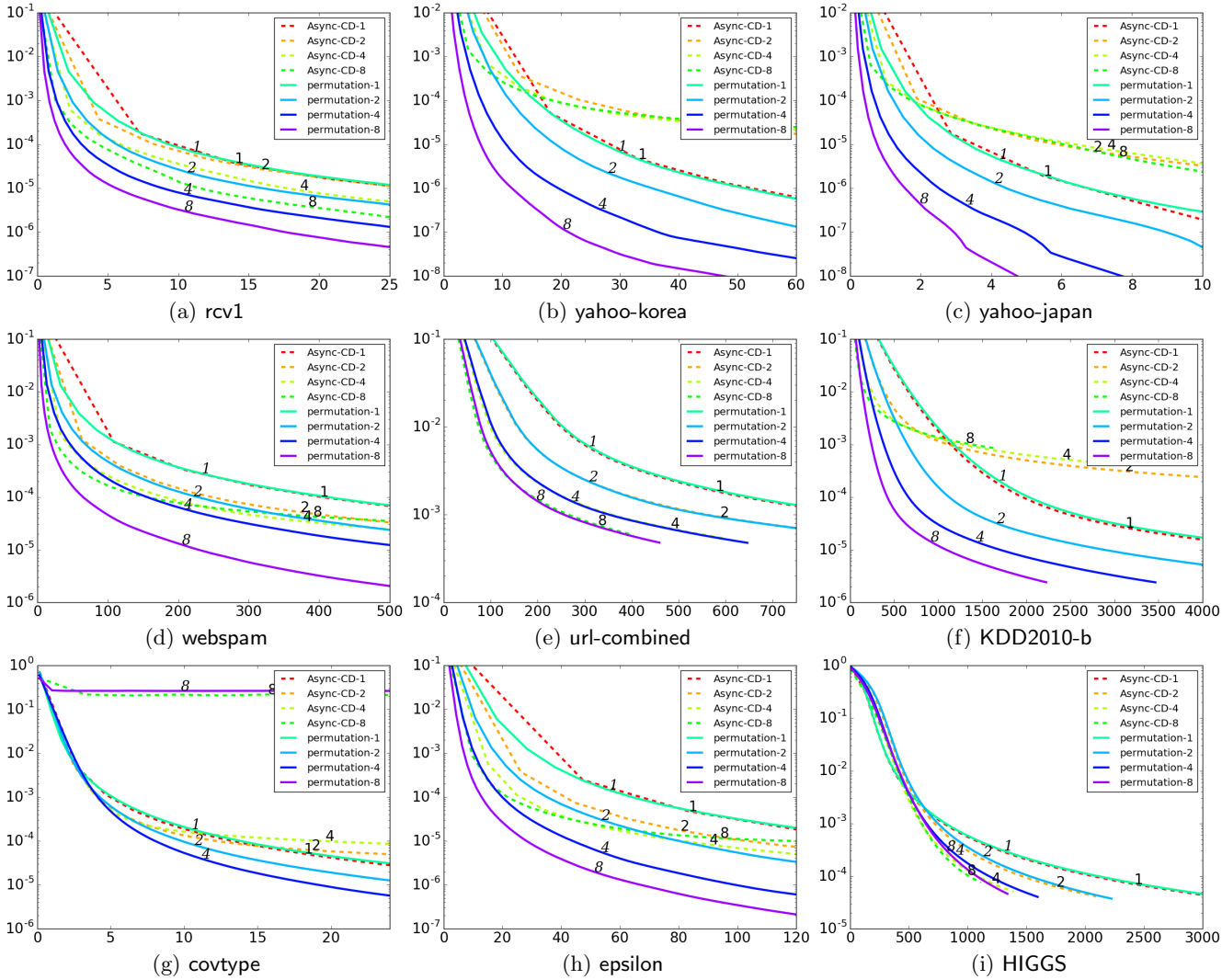


Figure IV: A comparison between two asynchronous dual CD implementations *without* applying the shrinking technique. "permutation" is our implementation that randomly permutes the indices in the beginning of each outer iteration, while the other is the experimental code from [2]. The $l1$ loss is considered.

convergence is from computing the gradient after resetting the set A. This calculation is parallelized in Algorithm I, so the issue of slow convergence becomes less significant.

We have also checked the situation when l2-loss is used. Results by using $\varepsilon = 0.1$ and 0.01 are respectively presented in Figures XIII and XIV. The improvement is less significant. The reason might be that shrinking is less effective for l2-loss SVM because α_i is now unbounded above.

References

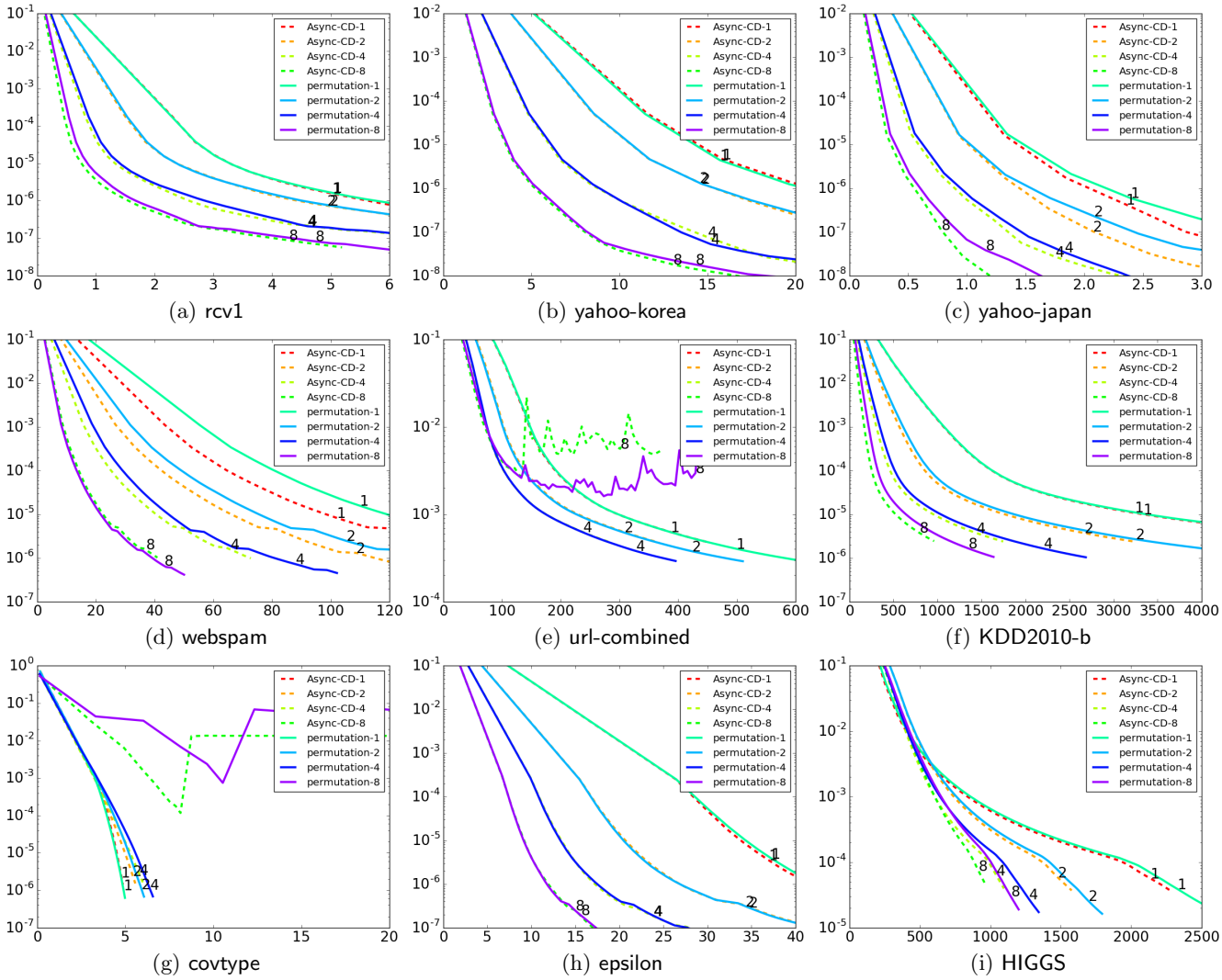
[1] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, 2008.

[2] C.-J. Hsieh, H.-F. Yu, and I. S. Dhillon. PASSCoDe: Parallel asynchronous stochastic dual coordinate descent. In *ICML*, 2015.

[3] M.-C. Lee, W.-L. Chiang, and C.-J. Lin. Fast matrix-vector multiplications for large-scale logistic regression on shared-memory systems. In *ICDM*, 2015.

[4] Z.-Q. Luo and P. Tseng. On the convergence of coordinate descent method for convex differentiable minimization. *J. Optim. Theory Appl.*, 72(1):7–35, 1992.

[5] M. Takáč, P. Richtárik, and N. Srebro. Distributed mini-batch SDCA, 2015. arXiv.

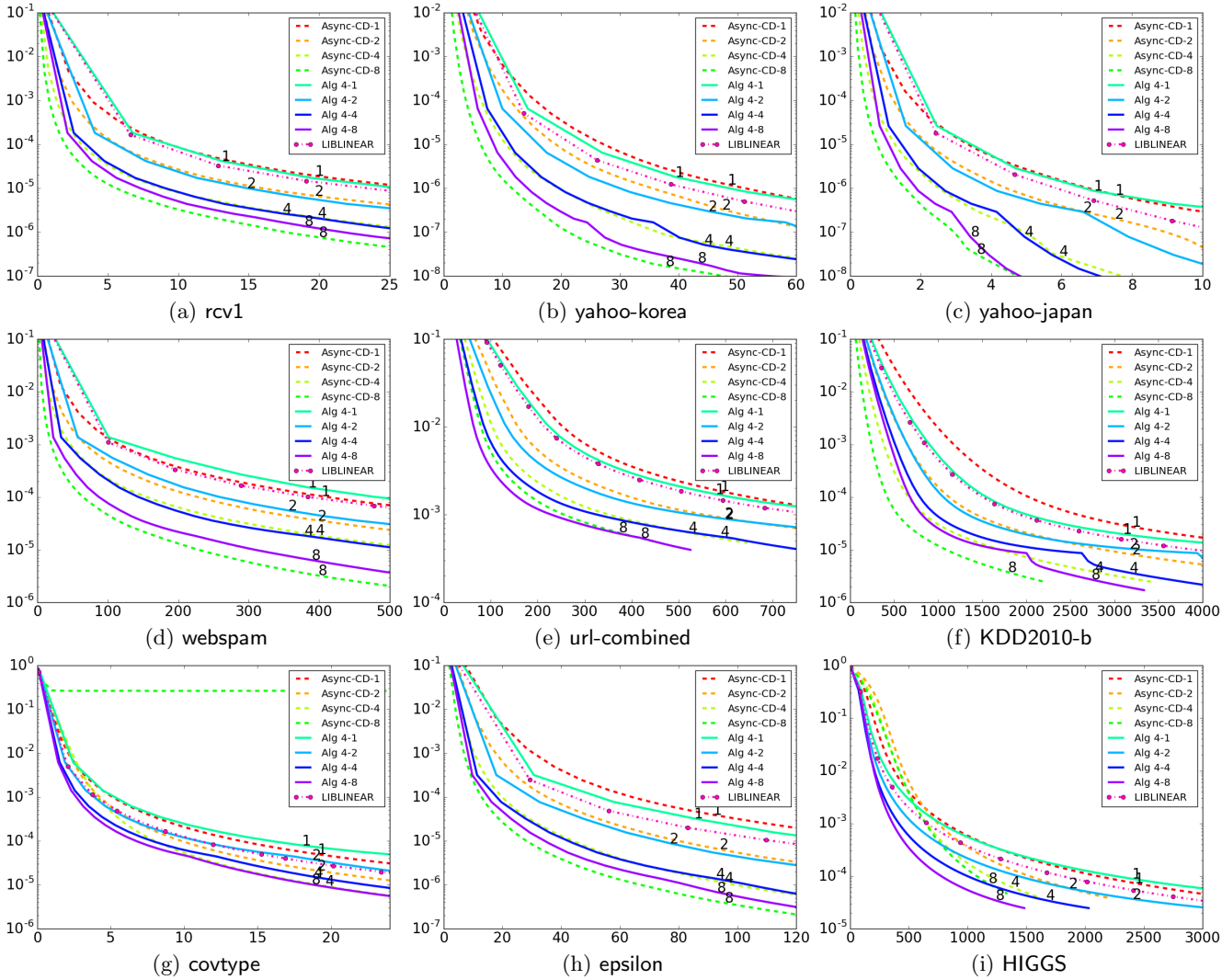


(a) rcv1 (b) yahoo-korea (c) yahoo-japan

(d) webspam (e) url-combined (f) KDD2010-b

(g) covtype (h) epsilon (i) HIGGS

Figure V: A comparison between two asynchronous dual CD implementations *with* applying the shrinking technique. "permutation" is our implementation that randomly permutes the indices in the beginning of each outer iteration, while the other is the experimental code from [2]. The $l1$ loss is considered.

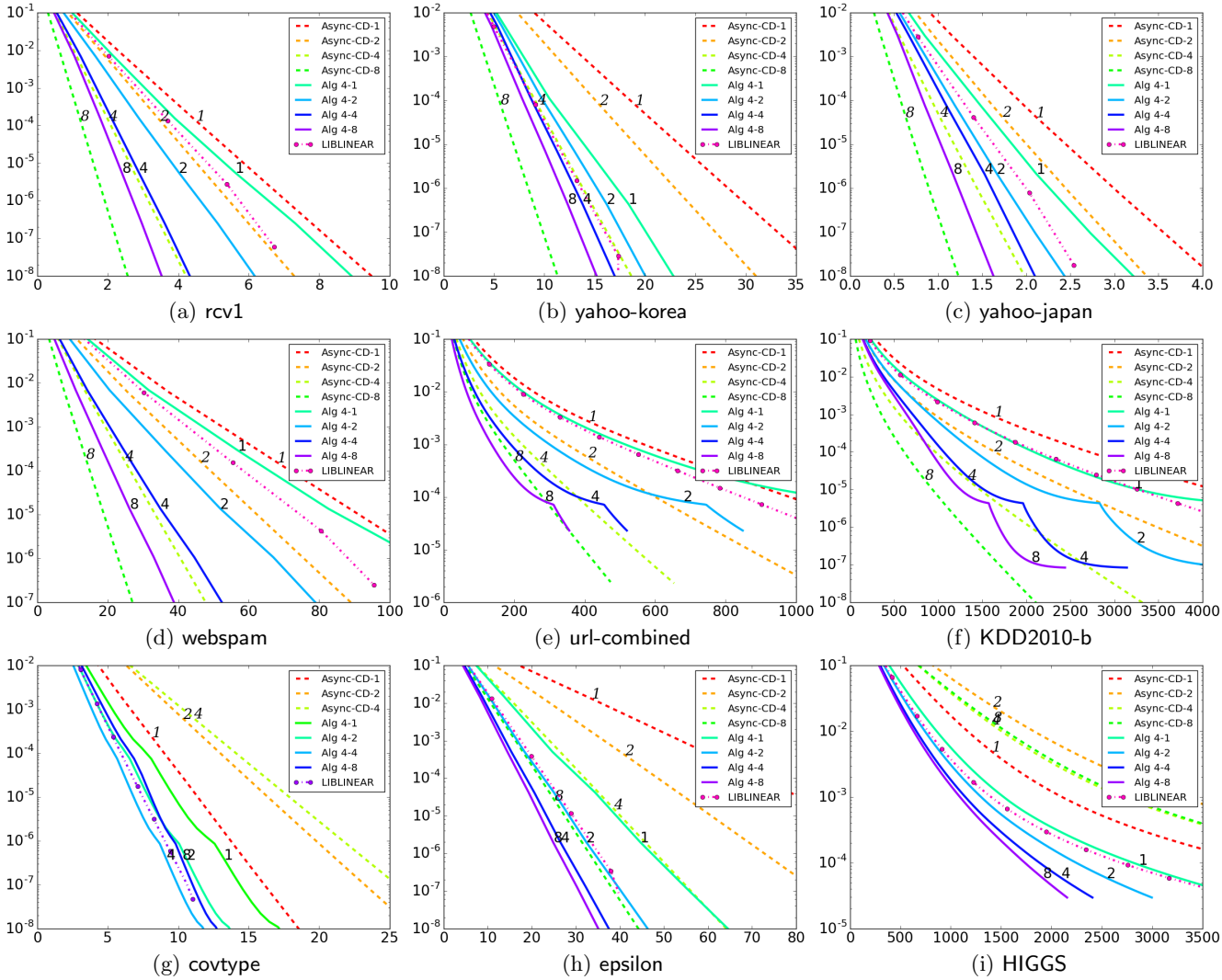


(a) rcv1 (b) yahoo-korea (c) yahoo-japan

(d) webspam (e) url-combined (f) KDD2010-b

(g) covtype (h) epsilon (i) HIGGS

Figure VI: A comparison of dual CD methods without applying the shrinking technique. All settings are the same as Figure 1. The $l1$ loss is considered.

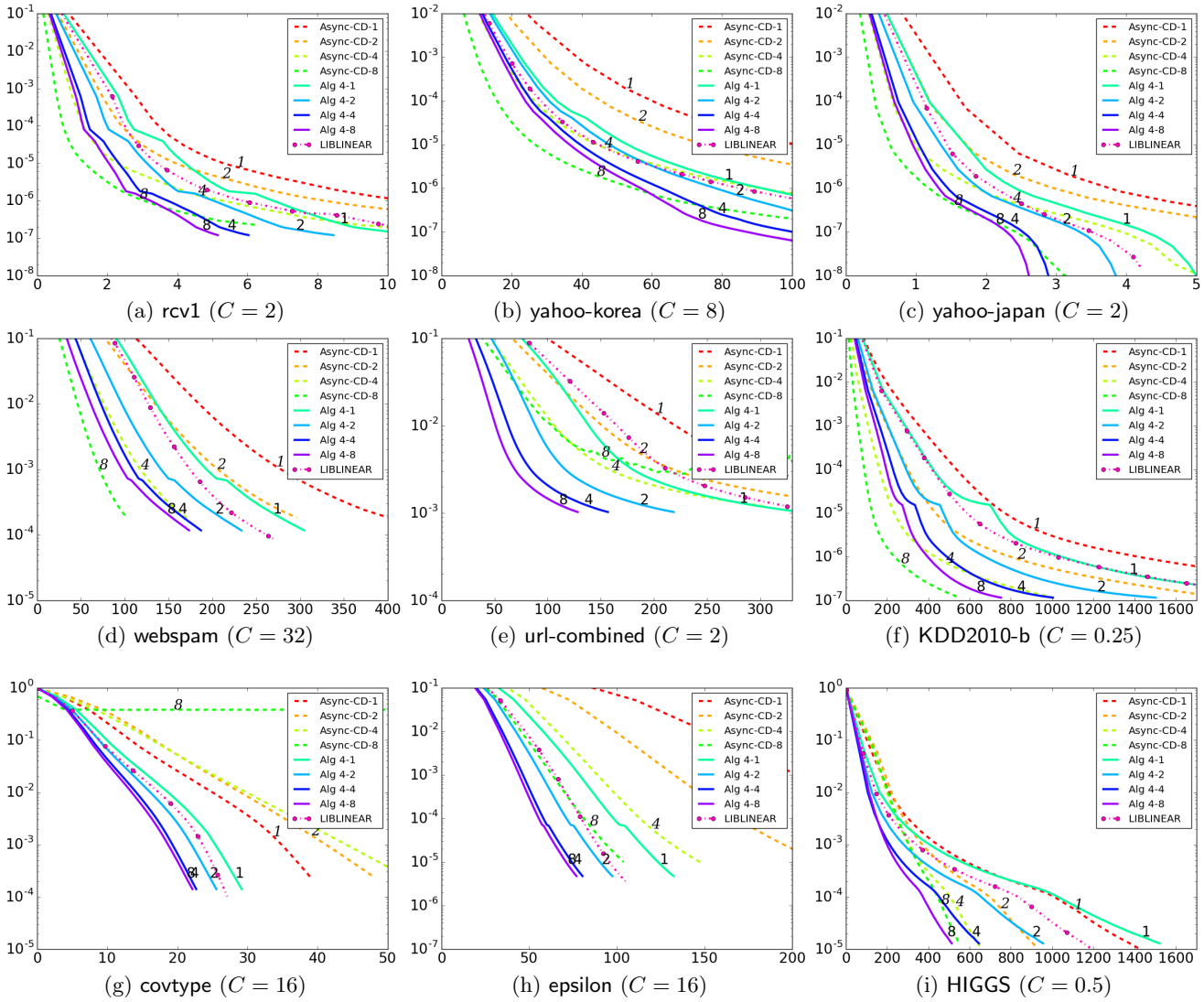


(a) rcv1 (b) yahoo-korea (c) yahoo-japan

(d) webspam (e) url-combined (f) KDD2010-b

(g) covtype (h) epsilon (i) HIGGS

Figure VII: A comparison of dual CD methods without applying the shrinking technique. All settings are the same as Figure 1. The $l2$ loss is considered.

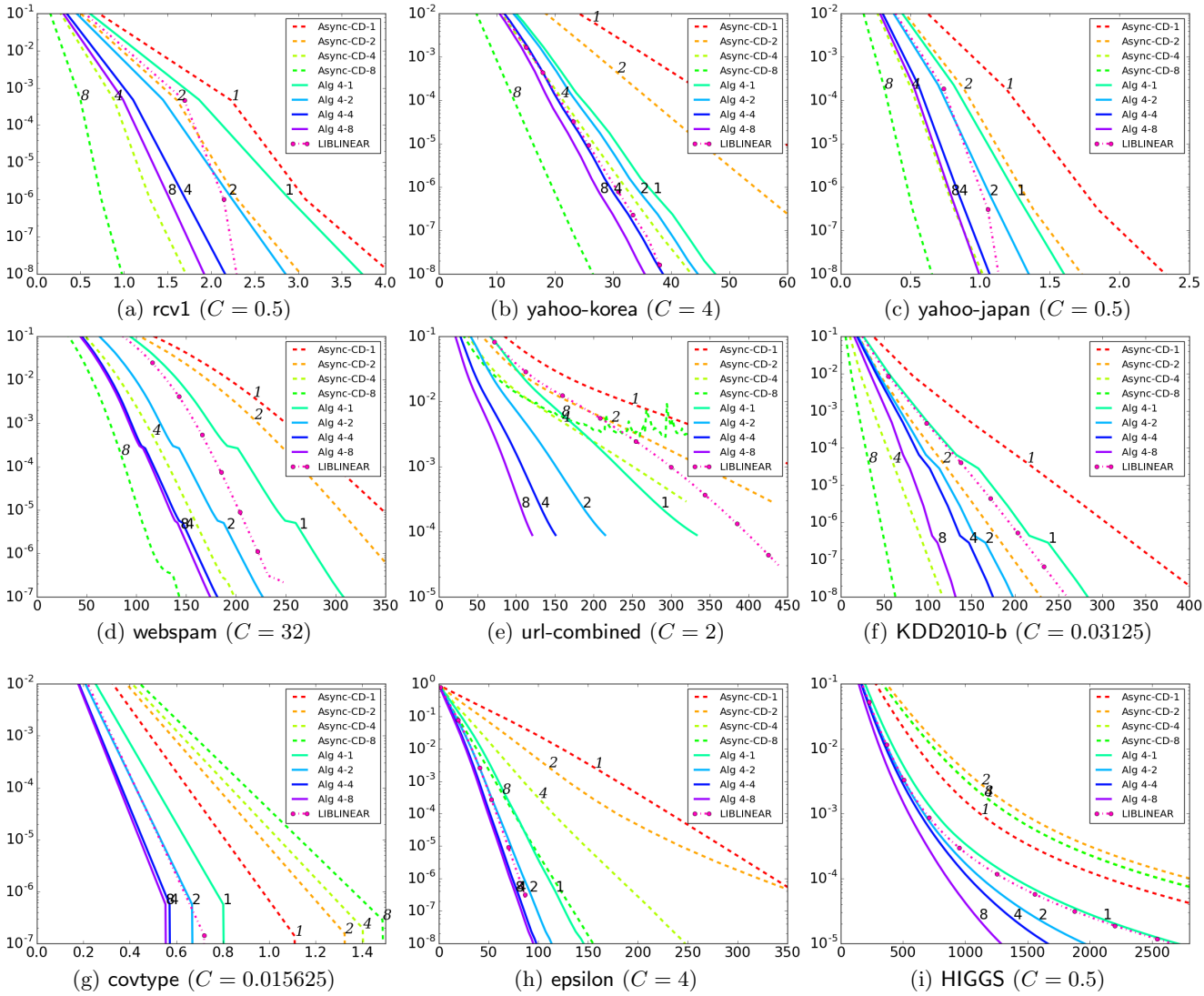


(a) rcv1 ($C = 2$) (b) yahoo-korea ($C = 8$) (c) yahoo-japan ($C = 2$)

(d) webspam ($C = 32$) (e) url-combined ($C = 2$) (f) KDD2010-b ($C = 0.25$)

(g) covtype ($C = 16$) (h) epsilon ($C = 16$) (i) HIGGS ($C = 0.5$)

Figure VIII: A comparison of different parallel dual CD methods. All settings are the same as Figure 1 except that the best C selected from cross validation is used; see the C value shown next to the data name. The $l1$ loss is used.

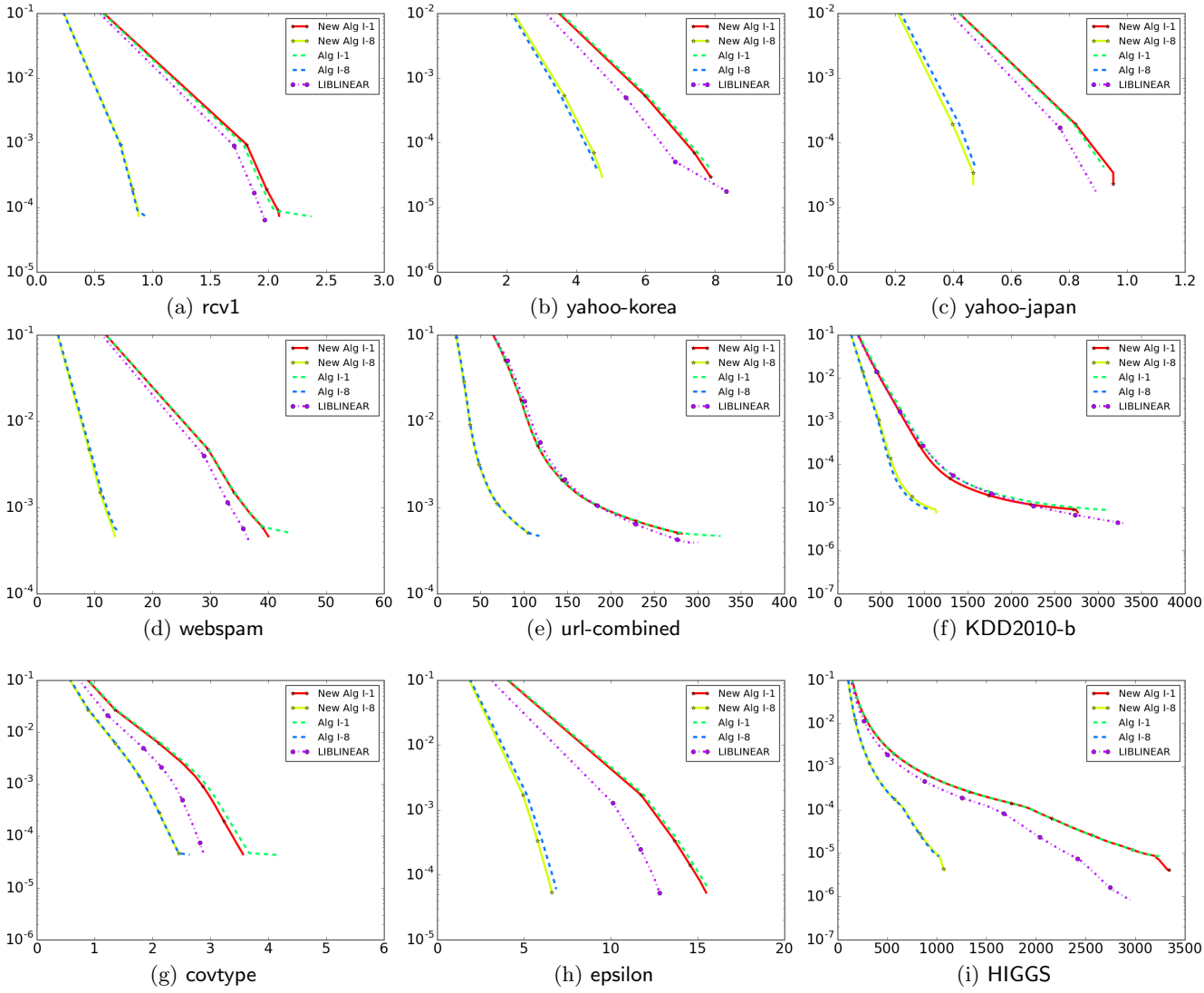


(a) rcv1 ($C = 0.5$) (b) yahoo-korea ($C = 4$) (c) yahoo-japan ($C = 0.5$)

(d) webspam ($C = 32$) (e) url-combined ($C = 2$) (f) KDD2010-b ($C = 0.03125$)

(g) covtype ($C = 0.015625$) (h) epsilon ($C = 4$) (i) HIGGS ($C = 0.5$)

Figure IX: A comparison of different parallel dual CD methods. All settings are the same as Figure 2 except that the best C selected from cross validation is used; see the C value shown next to the data name. The $l2$ loss is used.

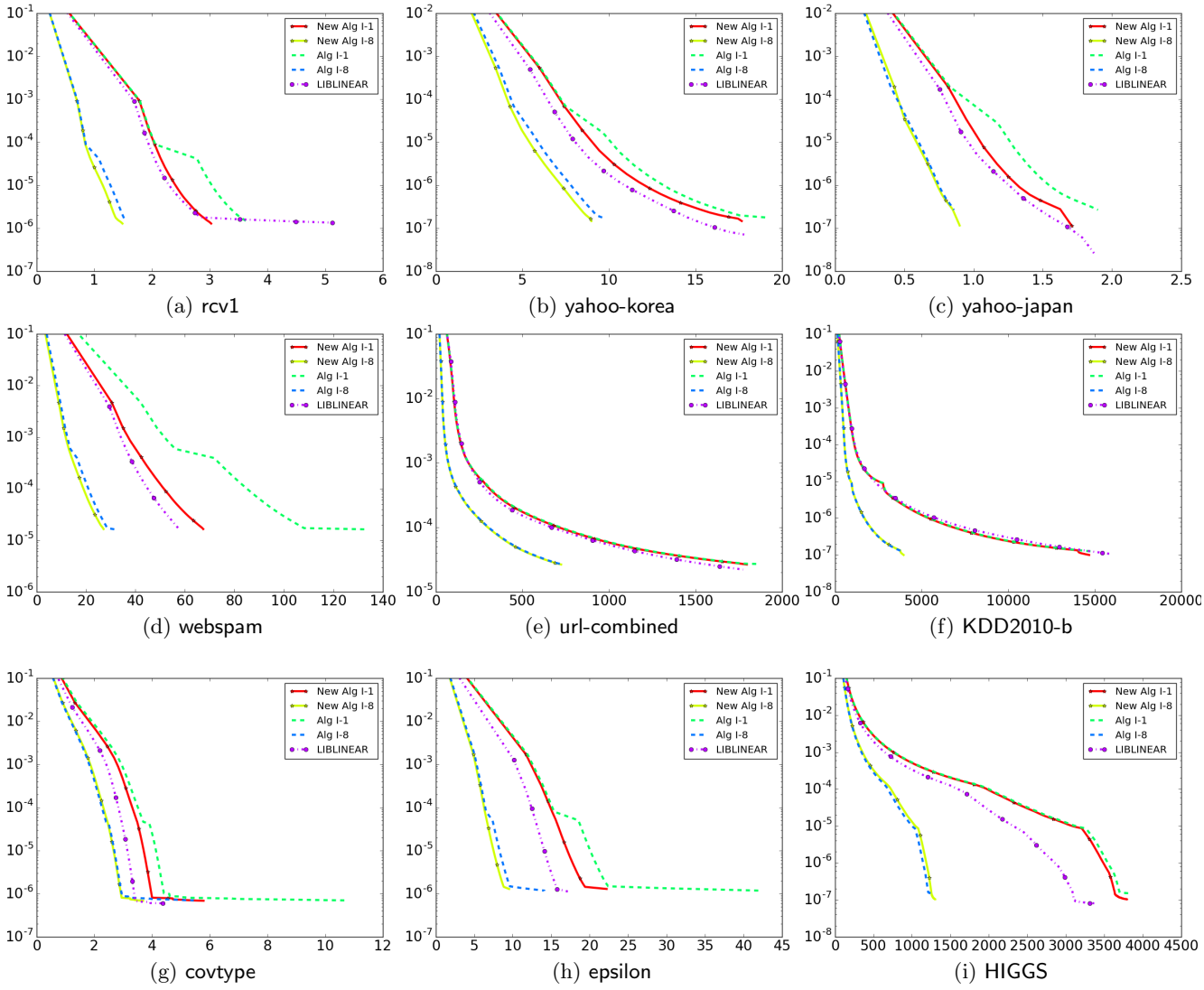


(a) rcv1 (b) yahoo-korea (c) yahoo-japan

(d) webspam (e) url-combined (f) KDD2010-b

(g) covtype (h) epsilon (i) HIGGS

Figure XI: A comparison between Algorithm I and the new setting (VII.1) **with** $\rho = 0.9$**. We set the stopping tolerance** $\varepsilon = 0.1$**. The** $l1$ **loss is used.**

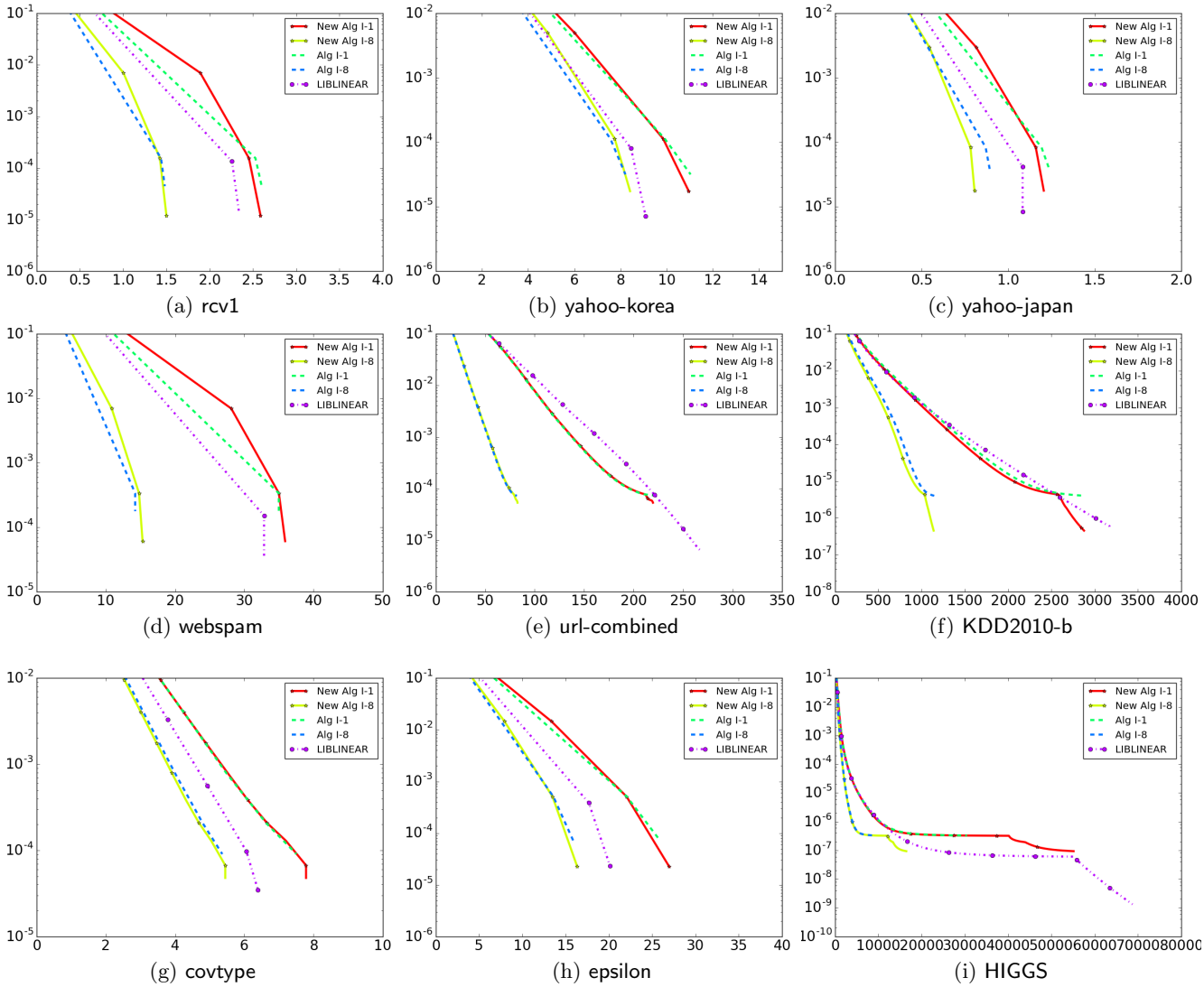


(a) rcv1 (b) yahoo-korea (c) yahoo-japan

(d) webspam (e) url-combined (f) KDD2010-b

(g) covtype (h) epsilon (i) HIGGS

Figure XII: A comparison between Algorithm I and the new setting (VII.1) **with $\rho = 0.9$. We set the stopping tolerance $\varepsilon = 0.01$. The $l1$ loss is used.**

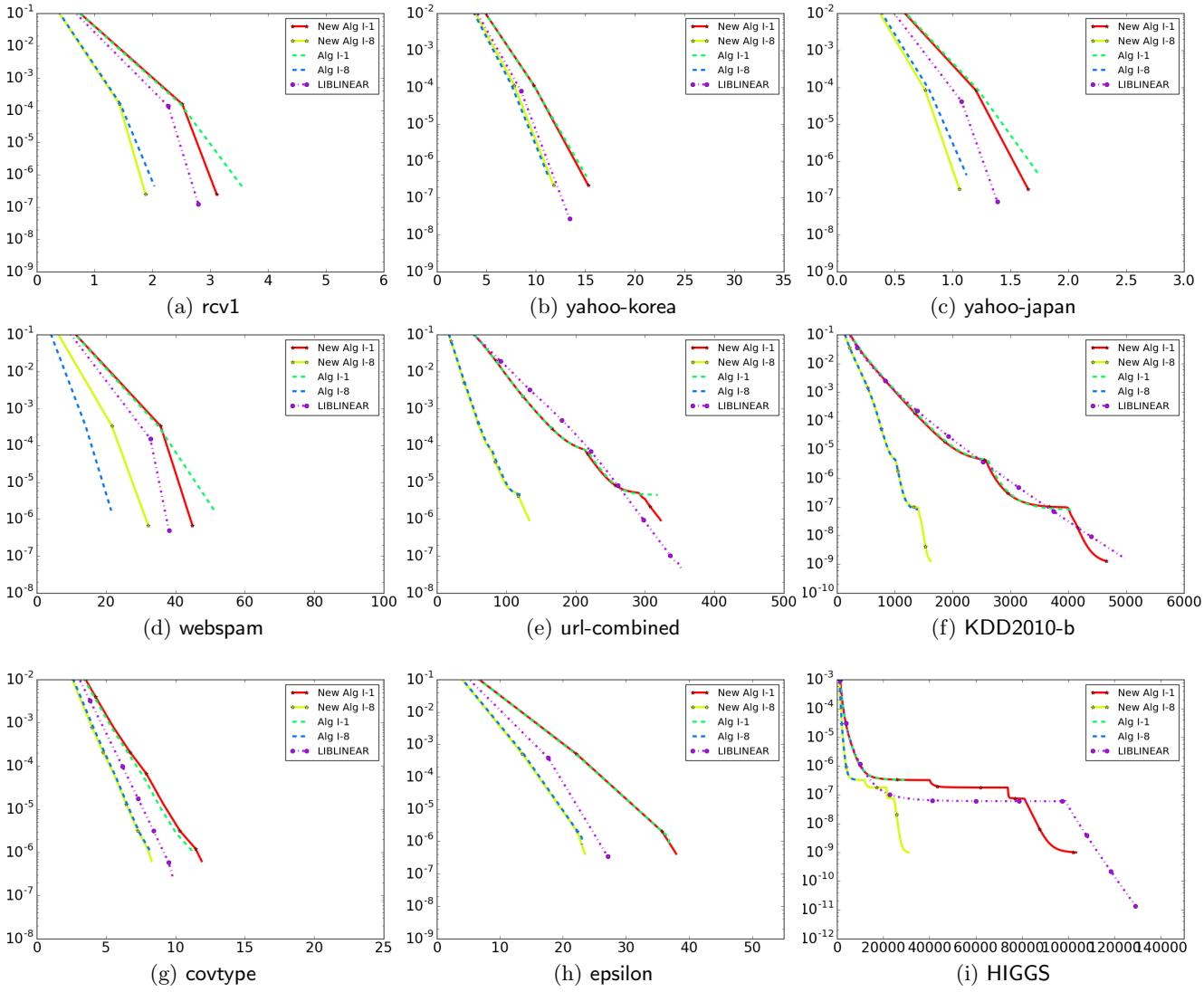


(a) rcv1 (b) yahoo-korea (c) yahoo-japan

(d) webspam (e) url-combined (f) KDD2010-b

(g) covtype (h) epsilon (i) HIGGS

Figure XIII: A comparison between Algorithm I and the new setting (VII.1) **with $\rho = 0.9$. We set the stopping tolerance $\varepsilon = 0.1$. The $l2$ loss is used.**



(a) rcv1 (b) yahoo-korea (c) yahoo-japan

(d) webspam (e) url-combined (f) KDD2010-b

(g) covtype (h) epsilon (i) HIGGS

Figure XIV: A comparison between Algorithm I and the new setting (VII.1) **with $\rho = 0.9$. We set the stopping tolerance $\varepsilon = 0.01$. The $l2$ loss is used.**