On the Equivalence of CoCoA+ and DisDCA

Ching-pei Lee
University of Illinois at Urbana-Champaign, 201 N. Goodwin Avenue, Urbana, IL 61801 USA

In this document, we show that the algorithm CoCoA+ (Ma et al., 2015) under the setting used in their experiments, which is also the best setting suggested by the authors that proposed this algorithm, is equivalent to the practical variant of DisDCA (Yang, 2013).

1. Notations

Consider the problems being solved, we first unify the notation in the two papers. Given training instances \( \{(x_i, y_i)\}_{i=1}^n \), the problem being solved are Problem (2) in Ma et al. (2015):

\[
\max_{\alpha \in \mathbb{R}^n} -\frac{1}{n} \sum_{j=1}^n l_j^* (-\alpha_j) - \frac{\lambda}{2} \| A\alpha \|^2, \quad (1)
\]

where \( A = [x_1, x_2, \ldots, x_n] \), and Problem (2) in Yang (2013):

\[
\max_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n -\phi_i^* (-\alpha_i) - \lambda g^* \left( \frac{1}{\lambda n} \sum_{i=1}^n \alpha_i x_i \right).
\]

By considering the special case \( g^* (\cdot) = \frac{1}{2} \| \cdot \|^2 \) and noting that \( \phi_i^* \) and \( l_j^* \) are just different notations for the same function, the two problems are equivalent. In the following analysis, we use the formulation of (1).

2. Algorithms

Assume there are \( K \) machines disjointly storing the training instances, the index set of the instances in machine \( k \) is defined as \( P_k \), and for any vector \( v \in \mathbb{R}^n \), define the vector \( v_{[k]} \in \mathbb{R}^n \) by

\[
(v_{[k]})_i = \begin{cases} 0, & \text{if } i \notin P_k; \\ v_i, & \text{if } i \in P_k. \end{cases}
\]

We list the CoCoA+ algorithm under the best setting (\( \sigma' = K, \gamma = 1 \)) suggested by the authors of Ma et al. (2015) in Algorithm 2, and list the algorithm of the practical variant of DisDCA in Algorithm 1. Note that for DisDCA with \( g(u) = \frac{1}{2} u^T u \), we have \( \nabla g(u) = u \).

Clearly, let

\[
\Delta \alpha = \delta, \quad u = w' + \frac{K}{\lambda n} A \Delta \alpha_{[k]},
\]

and observe that \( \alpha \) in Algorithm 2 is equivalent to \( \alpha + \alpha_{[k]} \) in Algorithm 2 by their update rules in line 1.2.3 of both algorithms, (2) and (3) are equivalent problems. Note that line 1.2.4 in Algorithm 2 indeed ensures \( u = w' + K \alpha/K \Delta \alpha_{[k]} \). Thus, the two algorithms are identical.

3. Implementation Comparison

Here we compare the codes CoCoA+ and Birds. CoCoA+ is the code released by the authors of Ma et al. (2015) implementing their algorithm in Apache Spark. As indicated in Ma et al. (2015), it is available in http://github.com/gingsmith/cocoa/Birds is the code released by the author of Yang (2013) implementing their practical variant of DisDCA proposed in that work using C++ and MPI. It is available at http://homepage.cs.uiowa.edu/~tyng/software.html.

We excerpt the core part of the codes solving the local subproblems to verify our argument in Section 2. Figure 1 shows lines 171-201 of the file CoCoA.scala in CoCoA+. Figure 2 shows lines 81-90 of the file inc_dual.cc in Birds.

Note that in Figure 1, the variable plus in CoCoA+ is true, and sigma is \( K \) as suggested in the paper. In Figure 2, the variable coeff is \( K/(\lambda n) \), and the variable mQ is the value of \( x_i^T x_i \).

In the beginning, \( \delta_{\text{grad}} \) in line 174 of CoCoA.scala is 0, so the value of \( \text{grad} \) after line 174 and the \( 1 - \text{prediction} \) part in line 84 of inc_dual.cc has the following relationship.

\[
\text{grad} = -\lambda n (1 - \text{prediction}) \quad (4)
\]

Lines 181-187 of CoCoA.scala and line 85 of inc_dual.cc are both projecting the variable back to a feasible region though the details are different.

From Line 188 of CoCoA.scala, we see that \( qii \) is \( K \) times of \( \text{Data.mQ(j)} \) in line 84 of inc_dual.cc. Combining these factors, we have that \(-\text{grad} / qii\) in line 191 of CoCoA.scala is the same as the first term of line 84 in inc_dual.cc.

\[
\text{grad} / qii = \frac{\left(y_i w^T x_i - 1\right) \lambda n}{x_i^T x_i K} = \frac{1 - y_i w^T x_i x_i^T x_i K}{x_i^T x_i \cdot K \lambda n} = (1 - \text{prediction}) / (\text{Data.mQ(j)} \ast \text{coeff})
\]
Algorithm 1 The CoCoA+ algorithm, under the setting suggested by the authors and also the setting used in the experiments of Ma et al. (2015).

- Input: number of iterations $T$, number of inner iterations $H$ for the local SDCA solver.
- Let $\alpha = 0$, $w^0 = 0$:
- For $t = 1, 2, \ldots, T$:
  1. Run the following process on the $K$ machines in parallel:
     1.1. Let $\Delta \alpha = 0$.
     1.2. For $h = 1, \ldots, H$:
        1.2.1. Pick $i \in P_k$ uniformly at random.
        1.2.2. Solve
        $$
        \delta_i^* = \arg \max_{\delta \in \mathbb{R}} -l_i^*(\alpha_i + \Delta \alpha_i) - (u^T A (\Delta \alpha_i + \delta e_i) - \frac{\lambda n K}{2} \frac{1}{\alpha_n} A (\Delta \alpha_i + \delta e_i)^2
        $$
        $$
        = \arg \max_{\delta \in \mathbb{R}} -l_i^* (-\alpha_i + \Delta \alpha_i) - (w^T + \frac{K}{\lambda n} A \Delta \alpha_i)^T x_i \delta - \frac{K}{2 \lambda n} ||x_i||^2 \delta^2 
        \tag{2}
        $$
        1.2.3. $\Delta \alpha_i \leftarrow \Delta \alpha_i + \delta_i^* e_i$.
        1.3. Update $\alpha_i \leftarrow \alpha_i + \Delta \alpha_i$.
  2. Obtain $w^{t+1} = w^t + \sum_{k=1}^K \frac{1}{\lambda_n} A \Delta \alpha_i$ on all machines.

Algorithm 2 The practical variant of the DisDCA algorithm.

- Input: number of iterations $T$, number of inner iterations $H$ for the local SDCA solver.
- Let $\alpha = 0$, $w^0 = 0$:
- For $t = 1, 2, \ldots, T$:
  1. Run the following process on the $K$ machines in parallel:
     1.1. Let $u = w^t$.
     1.2. For $h = 1, \ldots, H$:
        1.2.1. Pick $i \in P_k$ uniformly at random.
        1.2.2. Solve
        $$
        \Delta \alpha_i = \arg \max_{\Delta \alpha \in \mathbb{R}} -l_i^*(-\alpha_i - \Delta \alpha) - u^T x_i \Delta \alpha - \frac{K}{2 \lambda n} ||x_i||^2 (\Delta \alpha)^2
        \tag{3}
        $$
        1.2.3. $\alpha_i \leftarrow \alpha_i + \Delta \alpha$.
        1.2.4. $u = u + \frac{K}{\lambda_n} \Delta x_i \alpha$.
  2. Obtain $w^{t+1} = w^t + \sum_{k=1}^K \frac{1}{\lambda_n} A \Delta \alpha_i$ on all machines.
Then line 199 in Figure 1 and lines 87-88 in Figure 2 update the primal variables. Here a difference occurs. The update of WA in inc.dual.cc is $K$ times larger than the update of deltaW in CoCoA.scala. Thus, in the next round of inc.dual.cc, if we denote WA$_0$ as the value of WA in the previous round, and denote $K \ast$ deltaW as the vector being added to WA$_0$, we have that

$$Data.Xw(WA, j) = (WA_0 + K \ast \text{deltaW})^T x_j$$

which is exactly the same computation as line 174 of CoCoA.scala. Thus (4) still holds. Therefore, the algorithms behind the two implementations are identical.

**References**

Ma, Chenxin, Smith, Virginia, Jaggi, Martin, Jordan, Michael I, Richtárik, Peter, and Takáč, Martin. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the Thirty Second International Conference on Machine Learning*, 2015.

Yang, Tianbao. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems 26*, pp. 629–637, 2013.
// compute hinge loss gradient
val grad = {
  if (plus) {
    (y*(x.dot(w)+sigma*x.dot(deltaW)) - 1.0)*(lambda*n)
  } else {
    (y*(x.dot(w)) - 1.0)*(lambda*n)
  }
}

// compute projected gradient
var proj_grad = grad
if (alpha(idx) <= 0.0)
  proj_grad = Math.min(grad, 0)
else if (alpha(idx) >= 1.0)
  proj_grad = Math.max(grad, 0)
if (Math.abs(proj_grad) != 0.0) {
  var qii = if (plus) x.dot(x)*sigma else x.dot(x)
  var newAlpha = 1.0
  if (qii != 0.0) {
    newAlpha = Math.min(Math.max((alpha(idx) - (grad / qii)), 0.0), 1.0)
  }
}

// update primal and dual variables
val update = x.times(y*(newAlpha-alpha(idx))/(lambda*n))
if (!plus) {
  w = update.plus(w)
}
deltaW = update.plus(deltaW)
alpha(idx) = newAlpha

Figure 1. CoCoA.scala in CoCoA+

int is_class = (Data.y<int>(j)==myclass)?1:-1;
double prediction = is_class*(Data.Xw(WA, j));
double v = (1 - prediction)/(Data.nQ(j)*coeff) + Alpha[j];
v=std::max(std::min(v,1.0), 0.0);
double del_alpha=v- Alpha[j];
if (Data.fmt()=="dense") WA.add(Data.dX(j), del_alpha+is_class*coeff);
else
  WA.add(Data.sX(j), del_alpha+is_class*coeff);
phi[0] += del_alpha;
Alpha[j]=v;

Figure 2. inc_dual.cc in Birds