An Efficient Message Filtering Strategy Based on Asynchronous ADMM with L1 Regularization

Jiafeng Zhang
School of Computer Engineering and Science, Shanghai University, 200444, Shanghai, China
jiafengzhang@shu.edu.cn

Abstract. With the increment of data scale, distributed machine learning has received more and more attention. However, as the data grows, the dimension of the dataset will increase rapidly, which leads to the increment of the communication traffic in the distributed computing cluster and decreases the performance of the distributed algorithms. This paper proposes a message filtering strategy based on asynchronous alternating direction method of multipliers (ADMM), which can effectively reduce the communication time of the algorithm while ensuring the convergence of the algorithm. In this paper, a soft threshold filtering strategy based on L1 regularization is proposed to filter the parameter of master node, and a gradient truncation filtering strategy is proposed to filter the parameter of slave node. Besides, we update the algorithm asynchronously to reduce the waiting time of the master node. Experiments on large-scale sparse data show that our algorithm can effectively reduce the traffic of messages and make the algorithm reach convergence in a shorter time.

1. Introduction
In the era of big data, the scale of data is exponentially increasing, for machine learning, big data is not only reflected in the sample size of data, but also in the dimension of data. For example, in the problem of click-through rate prediction [1, 2] of online advertising, the dimensions of the dataset can reach more than 10 million dimensions, and the data is often stored in a sparse way. Under the large-scale data, it is difficult to optimize the traditional machine learning problem on a single machine. In this situation, distributed optimization [3, 4] is a better solution, and global optimization solution can be obtained by the cooperative computing and communication between nodes in a computing cluster.

ADMM [3] is a distributed optimization algorithm that can effectively solve global consensus optimization problems and has a wide range of applications in machine learning [5] and computer vision [6]. The ADMM algorithm accelerates the solution of machine learning problems by splitting the convex loss function $f(x)$ into $N$ sub-functions $f_i(x_i)$ and computing in parallel. The update of the distributed ADMM algorithm can be described as:

$$
\begin{align}
x_i^{k+1} &= \arg\min_{x_i} \left( f_i(x_i) + y_i^{kT}(x_i - z^k) + \left( \frac{\rho}{2} \right) \|x_i - z^k\|_2^2 \right) \\
z^{k+1} &= \frac{1}{N} \sum_{i=1}^{N} \left( x_i^{k+1} + \left( \frac{1}{\rho} \right) y_i^k \right) \\
y_i^{k+1} &= y_i^k + \rho(x_i^{k+1} - z^{k+1})
\end{align}
$$

1
Where \( k \) is the number of iterations of the ADMM algorithm and \( \rho \) is the penalty term parameter. In the distributed ADMM algorithm, there is a master node responsible for updating the global variable \( z \). The others are slave nodes, which solve the local objective function according to local dataset, obtain the local optimization solution \( x_{i}^{k+1} \) and \( y_{i}^{k+1} \). In each iteration, the master node need to collect all the local optimization solution \( x_{i}^{k+1} \) and \( y_{i}^{k+1} \) from slave nodes to update global variable \( z \). The ADMM algorithm achieves convergence in a certain number of iterations by alternating updates of the \( x, z \) and \( y \).

In the distributed ADMM algorithms, each iteration, the master node need to communicate with all the slave nodes. However, when the dimension of the communication parameters is high, the communication time in the network will be greatly increased. In addition, because of the computing performance of each slave node is different, and the master node must wait the slowest node to complete computing for each iteration, which will make the algorithm waste a lot of time on waiting.

There are many researches to solve the optimizations problems of large-scale sparse data [7, 8]. The main idea of them is that each dimension of the model parameter has different convergence speed. In their approaches, each iteration, only partial valid dimensions are selected to update the model parameter, and the algorithms achieve faster convergence. In addition, recently, asynchronous update mechanism [4, 9, 10] is also proposed to reduce the waiting time of the distributed algorithm.

Motivated by the above researches, this paper proposes an efficient message filtering strategy based on ADMM algorithm to solve the communication problem of distributed optimization in large-scale sparse dataset. In our strategy, message will be filtered before the parameter is sent, which reduces the message traffic and communication time is significantly reduced. In addition, we also adopted the asynchronous way to update the algorithm, and the waiting time is reduced. The experiment based on sparse logistic regression shows that our algorithm achieves better convergence and lower communicate latency compared with the algorithm without message filtering.

This paper is organized as follow. Section 2 surveys related works. Section 3 describes the proposed algorithm, including the message filtering strategy and the asynchronous update mechanism. Section 4 carries on the distributed experiments to the algorithm with large-scale datasets. Section 5 concludes the paper.

2. Related work

In order to improve the computational efficiency of distributed algorithms, many researches have focused on reducing communication time and reducing the latency of computing network. In this paper, we will review the ADMM algorithm with asynchronously update and the distributed algorithm with message filtering.

In 2014, R.L. Zhang [9] proposed an asynchronous ADMM algorithm to reduce the waiting time while ensuring the convergence of the algorithm. In their approach, algorithm can update asynchronously with two control conditions: partial barrier and bounded delays. Reference [11] extends the asynchronous ADMM algorithm to solve the problem of non-convex optimization and gives a detailed. Reference [12] proposed an ADMM algorithm based on parameter server architecture, which divides the data according to the model, the service node is responsible for the aggregation of the model parameters, and the working node performs the asynchronous update calculation. Although this algorithm divides the data into dimensions, and reduces the traffic of communication, but it also increases the complexity of the algorithm.

Li Mu [13] proposed an asynchronous distributed block proximal gradient algorithm, which can filter the message of network, and the algorithm set a bounded delay to make the algorithm update asynchronously. In their approach, several message filter strategies is proposed to compress the data, and reduce the communication time of the network. In addition, for the optimization algorithm of large-scale sparse data, [8] proposed the FOBOS algorithm, the idea of the algorithm is to set a gradient truncation, at each iteration, the value of a dimension in the gradient is little than the gradient truncation, the dimension will be filtered. In this way, the algorithm achieves faster convergence. And [7] proposed
a deep gradient compression algorithm to reduce the communication bandwidth for distributed training without losing accuracy

3. Algorithm Development

In this section, we will describe the message filtering strategy and the asynchronous update mechanism. Since the distributed ADMM update with a master node and several slave nodes. For master node, we propose a soft threshold filtering strategy to filter the message of global variable $z$. For slave node, we propose a gradient truncation update strategy to filter the message of local variable $x_i$ and $y_i$. For the asynchronous control of the algorithm, we refer to [9]the implementation of the asynchronous ADMM algorithm.

3.1. Message filtering strategy

In the distributed optimization with large-scale sparse dataset, the dimensions of the model parameters will be very high. Due to the different convergence speeds of different dimensions of the parameters, for those dimensions that have already converged, we can filter it before the message sent and reduce the communication traffic. In addition, when the large-scale sparse dataset is divided into several sub-datasets, some dimensions are invalid for a sub-dataset, and the value of these dimensions is 0 in the local model parameter. These dimensions have no benefit to the update of local model parameter, and we can also filter it to reduce communication.

Before introducing the message filtering strategy, we first define the message format for communication. The format of each message meta is defined as follows:

$$msg_i = (k, v)$$ (2)

Where $k$ denotes the index of each message in the parameter and $v$ denotes the value corresponding to the $k$th dimension of the parameter. As shown in figure 1, before the message is sent, the valid elements of the parameter will be collected with the index marked. Then the collected elements with indexes will be sent to receiver. After receiving the filtered message, the receiver needs to aggregate the message and put the valid elements into the related index in the message cache.

![Figure 1. Message filtering and aggregation.](image)

3.1.1. Soft threshold filtering strategy.

In the distributed ADMM algorithm, the master node is responsible for the update of the global variable $z$, and the update formula of $z$ is as shown in the equation (2), where $g(z)$ is generally a regularization term. When the ADMM algorithm contains the L1 regularization, $g(z) = \lambda \|z\|_1$, the update of $z$ can be reformulate as:

$$z^{k+1} = \min_z \left( \lambda \|z\|_1 + \sum_{i=1}^{N} \left( -y_i^k z + \frac{\rho}{2} \|x_i^{k+1} - z\|_2^2 \right) \right)$$ (3)

Since the update of $z$ needs to collect the variables from all the slave node, and the update of $z$ is quadratic equation, the update of $z$ can be simplified as:

$$z^{k+1} = \min_z \left( \lambda \|z\|_1 + \frac{\rho N}{2} \|z - \bar{x}^{k+1} - \frac{1}{\rho} \bar{y}^k\|_2^2 \right)$$ (4)

Where $N$ denotes the number of slave nodes, $\bar{x}^{k+1}$ denotes the average value of the local variable $x_i^{k+1}$, and $\bar{y}^k$ denotes the average value of the dual variable $y_i^k$. Although the L1 regularization term is non-differentiable, we can still solve it by Sub-gradient differentiation:
\[ z^{k+1} := S_{\lambda/\rho}(\bar{x}^{k+1} + (1/\rho)\bar{y}^{k}) \]  

(5)

Where \( S_{\lambda/\rho} \) denotes the soft threshold operator. We can get the final update formula of the global variable \( z \):

\[
(z^{k+1})_j = \begin{cases} 
(\bar{x}^{k+1})_j + (1/\rho)(\bar{y}^{k})_j - \frac{\lambda}{\rho} & \text{if } (\bar{x}^{k+1})_j + (1/\rho)(\bar{y}^{k})_j > \lambda/\rho \\
0 & \text{if } |(\bar{x}^{k+1})_j + (1/\rho)(\bar{y}^{k})_j| \leq \lambda/\rho \\
(\bar{x}^{k+1})_j + (1/\rho)(\bar{y}^{k})_j - \frac{\lambda}{\rho} & \text{if } (\bar{x}^{k+1})_j + (1/\rho)(\bar{y}^{k})_j < -\lambda/\rho 
\end{cases}
\]

(6)

We use \((z^{k+1})_j\) to denote the value of the \(j\)th dimension of the \(z^{k+1}\) parameter, which can be derived from the above update formula when \(|(\bar{x}^{k+1})_j + (1/\rho)(\bar{y}^{k})_j| \leq \lambda/\rho\), the value of \((z^{k+1})_j\) takes as 0. In this situation, we can deem that \((z^{k+1})_j\) have no help with update of slave nodes, so this dimension will be filtered.

The master node filters the updated global variable \(z\) according to the soft threshold strategy, and compress it according to the defined message format, and sends it to each slave node. After receiving the filtered message from master node, the slave node will decompress the message and store the global variable \(z^{k+1}\). Then the local variable \(x^{k+1}_i\) and the dual variable \(y^{k+1}_i\) will update with the global variable \(z^{k+1}\).

### 3.1.2. Gradient truncation strategy

In the distributed ADMM algorithm, each slave node is responsible for the update of the local variable \(x_i\) and the dual variable \(y_i\). The dual variable \(y_i\) can be regarded as an auxiliary variable that keeps the global variable and the local variable consistent. According to the global consensus constraint, when the algorithm reaches convergence, the optimization solution \((x_i^*, z^*)\) of the ADMM algorithm satisfies \(x_i^* - z^* = 0\), and the update formula for dual variables can be simplified to:

\[ y^{k+1}_i = y^k + \rho \Delta y_i \]  

(7)

Where \(\Delta y_i = x^{k+1}_i - z^k\), it can be seen as the gradient of the parameter \(y^{k+1}_i\), when the algorithm approaches convergence, \(\Delta y_i\) approaches to 0.

However, with the different convergence of each dimension in the parameter, the value of each dimension in \(\Delta y_i\) is different. We set a truncation value \(\theta\) of \(\Delta y_i\), when the absolute value of a dimension in \(\Delta y_i\) is little than the truncation value \(\theta\), we consider that the dimension of the parameter has converged, and it will be filtered before parameter is sent. According to equation (1), we describe the update of \(x_i\) and \(y_i\) with message filtering as follow:

\[
x_i^{k+1}_i = \arg\min_{x_i} \left( f_i(x_i) + y_i^{k+1}^i (x_i - z^k) + \left(\frac{\rho}{2}\right) \|x_i - z^k\|_2^2 \right)
\]

\[
y_i^{k+1}_i = y_i^k + \rho \left( x_i^{k+1}_i - z^k \right)
\]

\[
(x_i^{k+1})_j = \begin{cases} 
(x_i^{k+1})_j & \text{if } |(x_i^{k+1})_j - (z^k)_j| \geq (1/\sqrt{K})\theta \\
0 & \text{if } |(x_i^{k+1})_j - (z^k)_j| < (1/\sqrt{K})\theta 
\end{cases}
\]
\[
(y_i^{k+1})_j = \begin{cases} 
\frac{k_i^{k+1}}{2} y_i^{k+1}_j - (z^k)_j \geq \frac{1}{\sqrt{k}} \theta \\
0 & \frac{k_i^{k+1}}{2} y_i^{k+1}_j - (z^k)_j < \frac{1}{\sqrt{k}} \theta 
\end{cases}
\]

Where \(x_i^{k+1/2}\) and \(y_i^{k+1/2}\) are intermediate result of \(x_i\) and \(y_i\). \(x_i^{k+1}_j\) and \(y_i^{k+1}_j\) denote the \(j\)th dimension of \(x_i^{k+1}\) and \(y_i^{k+1}\). While [11] shows that ADMM algorithm achieves convergence with \(O(1/\sqrt{k})\), so we adjust the truncation as \(\frac{1}{\sqrt{k}} \theta\) dynamically. At each iteration, slave nodes first perform the update of \(x_i\) and \(y_i\), and parameters will be filtered by updating \(x_i^{k+1/2}\) and \(y_i^{k+1/2}\), then the parameter \(x_i^{k+1/2}\) and \(y_i^{k+1/2}\) will be sent to the master node.

### 3.2. Asynchronous update of the algorithm

In the synchronous ADMM algorithm, the master node needs to collect local variable \(x, y\) from all the slave nodes to update the global variable \(z\). It means that, each iteration, the master node needs to wait the update completed of the slowest node, and much time is wasted on waiting. In the asynchronous ADMM algorithm, the master node only collects local variable from partial slave nodes, and the slave nodes can update asynchronously in a bounded delay. In this way, the asynchronous ADMM algorithm can achieve convergence in a shorter time. Next, we will introduce the update method of asynchronous ADMM algorithm.

#### 3.2.1. Partial barrier

Partial barrier denotes that, each iteration, master node only collects local variable from partial slave nodes, and does not need to wait the slowest slave node to complete update. In this paper we use \(S\) to denote the partial barrier, and use \(\Phi\) as a node set to record the partial slave nodes which the master node received message from. In each iteration, only when the count of message master collected reach to \(S(|\Phi| \geq S)\), the master could to update global variable \(z\). After completing update of \(z\), the global variable \(z\) will be sent to the slave nodes which in \(\Phi\).

#### 3.2.2. Bounded delay

Under the control of partial barrier conditions, the slave nodes can update asynchronously. However, if each node performs the update completely asynchronously, the convergence of the ADMM algorithm will not be guaranteed. Therefore, bounded delay is set to control the update of slave nodes. We use \(\tau\) to denote the value of bounded delay. Bounded delay denotes that the master node must receive at least one message sent by each slave node every \(\tau\) iterations. In this way the convergence of the ADMM algorithm can be guaranteed. Specifically, the master node sets a count variable \(\tau_i\) for each slave node. When the master node receives the message from the node \(i\), the value of the corresponding count variable \(\tau_i\) in the master node is set to 1, when the \(z\) variable of the master has updated, the value of all the count variables \(\tau_i\) is incremented by 1. When the delay of a node reaches \(\tau (\tau_i > \tau)\), the other slave nodes must stop updating and wait for the node to complete the update.

We describe the message filtering strategy based on asynchronous update as shown in figure 2. In a network of one master node and three slave nodes, we set the partial barrier \(S\) to 2, and set bounded delay \(\tau\) to 10. As shown in the figure, at iteration 0, the master node receives the message from the node 1 and the slave node 3, the partial barrier reaches to 2. Then the master node can update the variable \(z\), and send the message to the node 1 and node 3 after message filtering. At this time, the delay count \(\tau_1\) and \(\tau_3\) will set to 1, and \(\tau_2\) will increment by 1 and set to 2. When \(\tau_2\) reaches to 10, slave node 1 and node 3 must stop update and wait slave node 2 to complete update. Based on the above analysis, we describe the asynchronous ADMM algorithm based on message filtering in algorithm 1.
Algorithm 1 Asynchronous ADMM algorithm based on message filtering

Algorithm of the Master node:
1: \( k = 0, \tau_i = 1, \) initialize \( x_i, y_i, \)  \( i = 1,2, ... N; \)
2: repeat
3: repeat
4: Waiting to receive a message, and store the slave nodes in \( \phi^k; \)
5: until \( |\phi^k| \geq S \) and \( \max(\tau_1, \tau_2 \ldots \tau_N) \leq \tau \)
6: for node \( i \in \phi^k: \) do
7: \( \tau_i = 1; \)
8: Decompress the message and aggregate the message into \( x_i \) and \( y_i; \)
9: end for
10: for node \( i \notin \phi^k: \) do
11: \( \tau_i = \tau_i + 1; \)
12: end for
13: Update \( z \) and filter the message according to equation(6);
14: for node \( i \in \phi^k: \) do
15: Send the filtered message to node \( i; \)
16: end for
17: Set \( k = k + 1; \)
18: until satisfies the stopping condition;

Algorithm of the Slave node:
1: Initialize \( x_i, y_i \) and \( z, \) set \( k = 0; \)
2: repeat
3: Waiting to receive message from master node, decompress the message and aggregate the message into \( z; \)
4: Update \( x_i, y_i \) and filter the message according to equation(8);
5: Send the filtered message to master node;
6: until satisfies the stopping condition;

4. Experiments
In this Section, we implement the ADMM algorithm based on message filtering strategy and use it to solve the sparse logistic regression problem under large-scale datasets. Logistic regression algorithm is
a classic machine learning algorithm. It is widely applied in data mining, click through rate prediction and other fields. The problem of logistic regression can be formulated as:

\[ L(W) = \sum_{i=1}^{N} [y_i(W \cdot x_i) - \log(1 + e^{W \cdot x_i})] + \frac{C}{2} \|W\|_2 \]  

(9)

Where \( W \) is the mode parameter, \( C \) is the penalty parameter, \( x_i \) and \( y_i \) are different from the variable of ADMM algorithm. In this situation, \( x_i \) is the sample of dataset, \( y_i \) is the label of sample.

The algorithm is implemented in C++, and MPICH is adopted for distributed computing and communication. We tested the algorithm on the high-performance cluster of Shanghai University and used 65 cores for computing. We use one core to simulate the master node and the other 64 cores are used to simulate the slave nodes. Each core is allocated with 4GB memory, and the communication bandwidth between nodes is 5.6Gb/s. We use the trust region newton method [14] to solve the optimization of \( W \), this method is seen to have a better performance than the method such as LBFGS [15] for large-scale logistic regression.

We perform experiment on 3 large-scale sparse datasets (kdd2010, avazu, url), which can be found in the Internet1. Among the datasets, kdd2010 has 19,264,097 samples and 1,163,024 dimensions, avazu has 40,428,967 samples and 1,000,000 dimensions, url has 2,396,130 samples and 3,231,961 dimensions.

We set the primal residual as \( r^k = \sum_{i=1}^{N} x_i^k - z^k \) according to the paper [3], where \( k \) denotes the algorithm \( k \)th iteration, \( N \) is the number of slave nodes, and then the convergence stopping condition of the algorithm can be described as:

\[ \|r^k\|_2 < \sqrt{N} \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max \left\{ \|x_i^k\|_2, \|z^k\|_2 \right\} \]  

(10)

Where \( \epsilon_{\text{abs}} > 0 \) is the absolute error, \( \epsilon_{\text{rel}} > 0 \) is the relative error, here we both set them to \( 10^{-3} \).

Figure 3. Compare the convergence of algorithm with different settings.

We first analyse the convergence performance of the algorithm with kdd2010 dataset. As shown in figure 3, we compare the primal residual of the algorithm under different settings. It can be seen from the figure that the convergence speed of the synchronization algorithm and the asynchronous algorithm are not much different when the message is not filtered, while the asynchronous algorithm adopting the message filtering strategy can achieve convergence in a shorter time. This is because of the message filtering strategy can significantly reduce the traffic of the message communication, and reduce the
communication time of the algorithm. Next, we will analyse the communication performance of the algorithm in detail.

We perform the experiment on the proposed algorithm with 3 different datasets. In this experiment, the penalty parameter $\rho$ of the ADMM is set to 2, and the penalty parameter $\lambda$ of L1 regularization is set to 1. When the algorithm updates with message filtering strategy, the gradient truncation $\theta$ is set as $10^{-3}$. When the algorithm updates asynchronously, partial barrier $S$ is set to 32, and bounded delay $\tau$ is set to 10.

Table 1. The detailed result under different dataset.

| Dataset | Synchronous algorithm without message filtering | Asynchronous algorithm with message filtering |
|---------|-----------------------------------------------|---------------------------------------------|
|         | Message traffic | C-time (s) | R-time (s) | Accuracy (%) | Message traffic | C-time (s) | R-time (s) | Accuracy (%) |
| avazu   | 1,000,000   | 91.45     | 122.38    | 80.17        | 11,797       | 24.45      | 56.34      | 78.90        |
| kdd2010 | 1,163,024  | 131.57    | 182.64    | 88.65        | 35,612       | 5.8        | 77.6       | 88.70        |
| url     | 3,231,961  | 607.35    | 827.22    | 98.01        | 48.231       | 11.16      | 104.03     | 97.81        |

Table 1 shows the detailed result of the proposed algorithm with different settings. Here, we use C-time to denote the communication time of algorithms, and use R-time to denote the run time of algorithms. As can be found in the table, the message traffic of each iteration can be greatly reduced while the algorithm adopting the message filtering strategy. Compared with synchronous algorithm, the Asynchronous algorithm with message filtering achieves faster convergence and has a better communication performance. When the algorithm is asynchronously updated, the algorithm has a lower accuracy. However, it is advisable to improve the performance of the algorithm with an appropriate accuracy loss.

5. Conclusion

This paper proposes a message filtering strategy based on asynchronous ADMM algorithm to solve the problem of communication in distributed optimization with large-scale sparse data. By using L1 regularity, a soft threshold filtering strategy is set on the master node, and gradient truncation is set by the update feature of the slave node. In addition, the algorithm adopts the asynchronous update mechanism to reduce the waiting time of the network. The related experiments show that the message filtering algorithm based on asynchronous ADMM can significantly reduce the communication time of the algorithm while ensuring the convergence of the algorithm. In the next work, the message filtering strategy will be considered to be applied to the ADMM algorithm of model parallel, and make efficiency of the algorithm will be further improved.

References

[1] Ho C H and Lin C J 2012 Journal of Machine Learning Research 13 3323-3348
[2] Canini K, Chandra T, Ie E, McFadden J, Goldman K, Gunter M, Harmsen J, LeFevre K, Lepikhin D, Linaretti T et al. 2012 Technical Talk 1 113
[3] Boyd S, Parikh N, Chu E, Peleato B, Eckstein J et al. 2011 Foundations and Trends in Machine learning 3 1-122
[4] Li M, Andersen D G and Smola A 2013 Distributed delayed proximal gradient methods NIPS Workshop on Optimization for Machine Learning vol 3 p 3
[5] Lubell-Doughtie P and Sondag J 2013 Practical distributed classification using the alternating direction method of multipliers algorithm 2013 IEEE International Conference on Big Data (IEEE) pp 773-776
[6] O’Connor D and Vandenberghe L 2014 SIAM Journal on Imaging Sciences 7 1724-1754
[7] Lin Y, Han S, Mao H, Wang Y and Dally W J 2017 Deep gradient compression: Reducing the communication bandwidth for distributed training 31st Conference on Neural Information Processing Systems pp 1-9
[8] Singer Y and Duchi J C 2009 Efficient learning using forward-backward splitting *Advances in Neural Information Processing Systems* pp 495-503

[9] Zhang R and Kwok J 2014 Asynchronous distributed admm for consensus optimization *International Conference on Machine Learning* pp 1701-1709

[10] Agarwal A and Duchi J C 2011 Distributed delayed stochastic optimization *Advances in Neural Information Processing Systems* pp 873-881

[11] Chang T H, Hong M, Liao W C and Wang X 2016 *IEEE Transactions on Signal Processing* 64 3118-3130

[12] Zhu R, Niu D and Li Z 2018 *arXiv preprint arXiv:1802.08882*

[13] Li M, Andersen D G, Smola A J and Yu K 2014 Communication efficient distributed machine learning with the parameter server *Advances in Neural Information Processing Systems* pp 19-27

[14] Lin C J, Weng R C and Keerthi S S 2007 Trust region newton methods for large-scale logistic regression *Proceedings of the 24th international conference on Machine learning* (ACM) pp 561-568

[15] Nocedal J 1980 *Mathematics of computation* 35 773-782