Non-linear Constrained Optimization Problem using Byzantine Distributed Optimization Algorithm

1st Ming-Yu Chung
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
r11921043@ntu.edu.tw

2nd Po-Yu Chen
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
r11921a15@ntu.edu.tw

***Abstract*—The constrained optimization problem is an important topic nowadays. An efficient algorithm of the constrained optimization algorithm is crucial in many fields, such as applied mathematics, signal processing, computer science, etc. One of the most famous algorithms is linearization method. However, we notice that the linearization method would produce terrible outcomes when some parts of functions or constraints in the optimization problem are unsuitable for linear approximation, especially for those who are non-linear. This would lead to a extremely small step size. To deal with this problem, in this report, we develop an algorithm based on linearization method and Byzantine distributed optimization algorithm to solve non-linear constrained optimization problems.**

Our algorithm proposed in this report is constructed as follows. We first introduce a novel idea. We extend the definition of "fault" in the Byzantine distributed optimization problem and then use the Byzantine distributed optimization algorithm (BDOA) as a filter. The BDOA would identify which functions or constraints is suitable for linear approximation and hence divide the optimization problem into two parts. For the part that constraints are suitable for linear approximation, we use linearization method; for the part that constraints are unsuitable for linear optimization, we use the proximal point method.

We also perform a series of experiments to demonstrate that our algorithm is more efficient compared to linearization method. The experimental results show that our algorithm is experimental better than linearization method.

***Index Terms*—Non-linear constrained optimization; constrained optimization; Distributed optimization; Approximate fault-tolerance; Distributed gradient-descent**

I. Introduction

Constrained optimization problem is an important and critical topic for many fields, like applied mathematics, signal processing, computer science, etc. Many problems in different fields can be transformed as a constrained optimization problem. Take support vector machine (SVM) for example. Given a function class $\mathcal{H} = \{f_\theta : \mathcal{X} \to \mathcal{Y} | \theta \in \Theta\}$, distribution $\mathcal{D}$, dataset $D = \{(x, y) | x \in \mathcal{X}, y \in \mathcal{Y}, (x, y) \sim \mathcal{D}\}$, a constrained function $C_i : \mathcal{H} \times (\mathcal{X}, \mathcal{Y}) \to \mathbb{R}$ where $i = 1 \sim n$. The support vector machine (SVM) problem can be expressed as an optimization problem. That is,

$$\begin{cases} \arg\min_{\theta\in\Theta} \|\theta\|^2 \\ \text{subject to } \mathbb{E}_{(x,y)\sim\mathcal{D}}\ C_i(f_\theta, (x, y)) \le 0,\ \forall i \in [n]. \end{cases} \tag{SVM}$$

Since the distribution $\mathcal{D}$ is unknown for almost all cases, we usually simulate (SVM) as follows:

$$\begin{cases} \arg\min_{\theta\in\Theta} \|\theta\|^2 \\ \text{subject to } \frac{1}{|D|} \sum_{(x,y)\in D} C_i(f_\theta, (x, y)) \le 0,\ \forall i \in [n]. \end{cases} \tag{SVM*}$$

By **law of large numbers**, we can ensure that the solution of (SVM*) would converge to (SVM) in probability as $|D| \to \infty$.

The optimization problem (SVM*) is a typical constrained optimization problem. There are so many optimization algorithms to solve such a constrained optimization problem. One of the most well-known algorithms is **linearization method** [1]. However, if some parts of the constraints are too complicated, especially for non-linear constraints, the linearization method may perform terribly. To deal with this problem, in this report, we develop a constrained optimization algorithm based on **linearization method** and **Byzantine distributed optimization algorithm**. The details of linearization method and Byzantine distributed optimization algorithm would be explained in the section II.

The main idea of our algorithm is that we want to utilize the Byzantine distributed optimization algorithm to filter the constraints which are unsuitable for linear approximation. For the constraints suitable for linear approximation, we use linearization method; for the constraints unsuitable for linear optimization, we use the proximal point method. We expect that our method would perform better than linearization method for the cases where some constraints are non-linear.

Following are our main contributions in this report:

- We extend the definition of "fault" in the Byzantine distributed optimization problem. Then, we utilize the Byzantine distributed optimization algorithm as a filter to identify the constraints which are unsuitable for linear approximation.
- We develop an algorithm based on linearization method and Byzantine distributed optimization algorithm to deal with non-linear constrained optimization problems. We expect our algorithm to solve the problem more efficiently than linearization method.
- We perform some experiments to demonstrate the advantage of our algorithm compared with linearization

method.

II. Background knowledge and related works

In this section, we would first talk about previous works. Then, we would briefly introduce **Byzantine distributed optimization problem** and **Linearization method** which are important components of this report. Besides, we would also define the **constrained optimization problem** where we are interested in this report.

A. Previous works

In this report, what we are doing is hybrid two kinds of optimization algorithms to generate a better one. The most similar works are [2], [3]. The first one talks about how to hybrid two kinds of algorithms, an **improved genetic algorithm (IGA)** and an **improved particle swarm optimization (IPSO)**. The second one talks about how to apply Byzantine distributed optimizations to solve constrained optimization problems. However, no one ever researches how to hybrid Byzantine distributed optimization algorithm and constrained optimization algorithm to develop a more efficient constrained optimization algorithm. So, what we are doing is a very novel topic.

B. Byzantine distributed optimization problem

The Byzantine distributed optimization problem which we consider in this report can be described as following settings:

- **Agent**: There are m agents. For i-th agent, he holds a cost function $C_i : \mathbb{R}^n \to \mathbb{R}$. The i-th agent will also send the information of his cost function to the central server according to the order from the central server. We say the i-th agent is a Byzantine faulty agent if he sends incorrect information to the central server. Otherwise, we call him an honest agent.
- **Central server:** Central server aims to utilize the information obtained from each agent to solve the following optimization problem:

$$\arg\min_{x\in\mathbb{R}^n} \sum_{i\in\mathcal{H}} C_i(x), \tag{1}$$

  where $\mathcal{H}$ is the index set of honest agents and $\mathcal{B}$ is the index set of Byzantine faulty agents.

Byzantine distributed optimization problem is defined as the optimization problem (1) under the above settings. It is worth to mention that [4]–[7] have shown that *"The exact fault tolerance problem* (1) *cannot be solved without some specific condition (*$2\mathcal{B}$*-redundancy)"*. The $2\mathcal{B}$-redundancy is defined as following:

Definition 1 ($2\mathcal{B}$*-redundancy):* We say that the Byzantine distributed optimization problem satisfies $2\mathcal{B}$-redundancy if the number of the honest agents is two times more than the number of the Byzantine faulty agents. That is,

$$|\mathcal{H}| \geq 2|\mathcal{B}|. \tag{2}$$

Hence, in this project, we always assume $2\mathcal{B}$-redundancy is satisfied.

On the other hand, the algorithm solving Byzantine distributed problem is called **Byzantine distributed optimization algorithm (BDOA)**. One of the most famous BDOA is **Gradient-Filter-based Distributed Gradient Descent**, whose key idea is to mitigate the detrimental impact of incorrect gradients. Following is the related works of this kind of BDOA: comparative gradient elimination (CGE) [8], coordinate-wise trimmed mean (CWTM) [4], geometric median-of-means (GMoM) [9].

C. Setting of the constrained optimization problem and our goal

After defining the Byzantine distributed optimization problem, we start to define the constrained optimization problem, which we consider in this report. In this project, we want to develop an algorithm to solve the following constrained optimization problem:

$$\begin{cases} \arg\min_{x\in\mathbb{R}^n} & C_0(x) \\ \text{subject to} & C_i(x) \leq 0, \ \text{where } i \in [m], \end{cases} \tag{P0}$$

where $C_i(x) : \mathbb{R}^n \to \mathbb{R}$, for any $i = 0 \sim m$.

In a practical situation, we would approximate (P0) as:

$$\begin{cases} \arg\min_{x\in\mathbb{R}^n} \quad C_0(x) + \sum_{i\in[m]} k \cdot \mathrm{ReLU}(C_i(x)), \end{cases} \tag{P0*}$$

where $k > 0$ and the solution of (P0*) tends to the solution of (P0) as $k \to \infty$. Notice that (P0*) is similar to the form of Byzantine distributed optimization problem (1). For convenience, in this report, we would not distinguish (P0*) and (P0) and utilize (P0*) to relate the BDOA.

In this report, we utilize the technique of BDOA to enhance the performance of some existing constrained optimization algorithms on (P0). To be more specific, we want to improve linearization method. [1]. The details of linearization method would be discussed in the next subsection.

D. Linearization method

As we have mentioned, linearization method is an algorithm for constrained optimization problem (P0). In this subsection, we would introduce the linearization method in detail according to [1]. The convergence analysis is not included in this report but the corresponding proof can be found in [1].

Just like the literal meaning, the main idea of linearization method is " linearize all functions in the constrained optimization problem (P0)". Given a smooth function $f : \mathbb{R}^n \to \mathbb{R}$. According to the Tyler expansion, we can express f as a linear approximation as follows:

$$f(x+p) = f(x) + \langle \nabla f(x), p\rangle + O(\|p\|^2), \tag{3}$$

where p is some unit direction in $\mathbb{R}^n$. Based on this, at first, we utilize (3) to linearize the constrained optimization problem (P0). We can obtain follows:

$$\begin{cases} \arg\min_p & C_0(x) + \langle \nabla C_0(x), p\rangle \\ \text{subject to} & C_i(x) + \langle \nabla C_i(x), p\rangle \leq 0, \ \text{where } i \in [m]. \end{cases}$$

Notice that the linear approximation is only valid when $\|p\|^2$ is sufficiently small. We must add some regularization terms in the optimization problem to avoid $\|p\|^2$ being too large. That is, we transform the optimization problem as follows:

$$\begin{cases} \arg\min_p & C_0(x) + \langle \nabla C_0(x), p \rangle + \|p\|^2 \\ \text{subject to} & C_i(x) + \langle \nabla C_i(x), p \rangle \leq 0, \forall i \in [m]. \end{cases} \tag{P1}$$

Then, what linearization method does is solve (P1) iteratively to obtain the solution of (P0), which can be expressed as **Algorithm** 1.

Algorithm 1: Linearization method

```
Given an initial point $x^0 \in \mathbb{R}^n$, $\epsilon > 0$, $N > 0$ and
  iteration number $I$;
for $k := 0$ to $I$ do
    $p^k \leftarrow$ the solution of (P1) at $x = x^k$;
    $\alpha^k \leftarrow \alpha(x^k, p^k, \epsilon, N)$;
    // $\alpha$ is defined at the bottom
    $x^{k+1} \leftarrow x^k + \alpha^k \cdot p^k$;
    $k \leftarrow k + 1$;
end
return $x^k$

// The function $\alpha(x^k, p^k, \epsilon, N)$ is defined
   as the minimum $i$ such that
   following inequality holds:
       $C_0(x^k + (\frac{1}{2})^i p^k) + NF(x^k + (\frac{1}{2})^i p^k) \leq$
           $C_0(x^k) + NF(x^k) - (\frac{1}{2})^i \epsilon \|p^k\|^2,$
   where $F(x) = \max_{i \in \{0,1,\ldots,m\}} C_i(x)$.
```

When the functions $C_i(x)$ have some good properties like convex, Lipschitz continuous, etc, we can ensure the output of **Algorithm** 1 would converge to the solution of the optimization problem (P0). More details and the convergence analysis are also discussed in the [1].

III. Proposed Algorithm

In the previous section, we briefly introduced the critical components of our proposed algorithm. Now, in this section, we would explain how to develop an algorithm for non-linear constrained optimization problems (P0), which is based on **linearization mehod** and **Byzantine distributed optimization algorithm (BDOA)**.

A. Observation on linearization method

First of all, we observe the linearization method (**Algorithm** 1). There are two important values in the algorithm, descent direction p^k and step-size α^k. When some parts of the functions $C_i(x)$ in (P0) are not suitable for linear approximation, the α^k will be very close to zero. Then, the linearization method would perform inefficiently due to

$$x^k + \alpha^k \cdot p^k \approx x^k + 0.$$

Inspired by this drawback of the linearization method, we figure out a method to cope with this problem of linearization method. We think that we should divide (P0) into two parts. For the part which is suitable for linear approximation, we use the linearization method to optimize; For the part where linear approximation performs terribly, we use another optimization algorithm. Thus, in order to realize our idea, we must need to construct a filter to identify which function $C_i(x)$ is suitable for linear approximation. In the next subsection, we will explain how we achieve this goal by BDOA.

B. Using the BDOA as a filter

Recall the definition of the Byzantine distributed optimization problem (1), we notice that BDOA can only optimize on the $C_i(x)$ which belongs to honest agents. The BDOA performs as a filter that identifies the index set of honest agents $\mathcal{H}$ and then only optimizes on the corresponding $C_i(x)$. Hence, if we can consider "unsuitable for linear approximation" as a fault in Byzantine distributed optimization problem, the BDOA will identify the index set of honest agents $\mathcal{H}$ automatically during the process of optimization.

Given a smooth function $f : \mathbb{R}^n \to \mathbb{R}$. Through the Taylor expansion again, we have

$$f(x+p) = f(x) + \langle \nabla f(x), p \rangle + \langle \nabla^2 f(x) p, p \rangle + O(\|p\|^3), \tag{4}$$

where $\nabla^2 f(x)$ is the hessian matrix of function f at x. From (4), we know that if $\| \langle \nabla^2 f(x) p, p \rangle \| >> 0$, the function f will be unsuitable for linear approximation since $\langle \nabla^2 f(x) p, p \rangle$ can not be ignored. Thus, it is reasonable to define the "unsuitable for linear approximation" as follows:

Definition 2 (N_{filter} unsuitable for linear approximation): Given a smooth function $f : \mathbb{R}^n \to \mathbb{R}$. We say f is N_{filter} unsuitable for linear approximation at x along with the direction p if

$$\| \langle \nabla^2 f(x) p, p \rangle \| > N_{\text{filter}}, \tag{5}$$

where $N_{\text{filter}} > 0$ is chosen arbitrarily.

After defining the "unsuitable for linear approximation" and considering it as a fault in Byzantine distributed optimization problem, we perform linearization method and **Gradient-Filter-based Distributed Gradient Descent**, one of BDOA, to achieve our goal. That is, we solve (P1) as following

1) We first solve (P1) at x^k and find the descent direction p^k.
2) Given $N_{\text{filter}} > 0$. We identify the index set of honest agents. To explain, we consider
$$\mathcal{H}' := \{ i | \| \langle \nabla^2 C_i(x^k) p^k, p^k \rangle \| \leq N_{\text{filter}}, \} \tag{6}$$
3) Define optimization problem (P1')
$$\begin{cases} \arg\min_{x \in \mathbb{R}^n} & C_0(x) \\ \text{subject to} & C_i(x) \leq 0, \text{ where } i \in \mathcal{H}', \end{cases} \tag{P1'}$$
4) Solve (P1') at x^k and find the descent direction p^k.
5) Return p^k.

By the definition of Byzantine distributed optimization problems, if we utilize linearization method **(Algorithm 1)** with the above method to solve (P1), we can ensure that the output of **Algorithm 1** will converge to the solution of the below optimization problem as $I \to \infty$:

$$\Big\{ \arg\min_{x \in \mathbb{R}^n} C_0(x) + \textstyle\sum_{i \in \mathcal{H}} k_b \cdot \mathrm{ReLU}(C_i(x)). \tag{PB*}$$

As we have mentioned, (PB*) is equivalent to (PB), which is defined as follows:

$$\begin{cases} \arg\min_{x \in \mathbb{R}^n} & C_0(x) \\ \text{subject to} & C_i(x) \le 0, \ \text{where } i \in \mathcal{H}, \end{cases} \tag{PB}$$

where $\mathcal{H}$ is the index set of the honest agents. Since we define “unsuitable for linear approximation” as a fault, $\mathcal{H}$ is equivalent to the index set that $C_i(x)$ is suitable for linear approximation.

To sum up, in this subsection, we have constructed an optimization algorithm to solve (PB) and identify the functions $C_i(x)$ which are unsuitable for linear approximation.

C. Ancillary optimization problem

In the previous subsection, we have developed an algorithm to solve (PB). Now, we turn to construct an algorithm to solve the ancillary optimization problem (PC), which is defined as follows:

$$\begin{cases} \arg\min_{x \in \mathbb{R}^n} & C_0(x) \\ \text{subject to} & C_i(x) \le 0, \ \text{where } i \in \mathcal{B}, \end{cases} \tag{PC}$$

The algorithm to achieve our goal is similar to the previous subsection. We consider follows:

1) We first solve (P1) at x^k and find the descent direction p^k.
2) Given $N_{\text{filter}} > 0$. We identify the index set of honest agents. To explain, we consider
$$\mathcal{B}' := \{ i | \| \langle \nabla^2 C_i(x^k) p^k, p^k \rangle \| > N_{\text{filter}}, \} \tag{7}$$
3) Use the **proximal point method** [10] to solve following optimization problem:
$$\Big\{ \arg\min_{x \in \mathbb{R}^n} C_0(x) + \textstyle\sum_{i \in \mathcal{B}'} k_c \cdot \mathrm{ReLU}(C_i(x)). \tag{P1”}$$
4) Return x^{k+1}, which is the solution of (P1”).

The x^k deduced by the above algorithm will converge to the solution of (PC*)

$$\Big\{ \arg\min_{x \in \mathbb{R}^n} \quad C_0(x) + \textstyle\sum_{i \in \mathcal{B}} k_c \cdot \mathrm{ReLU}(C_i(x)), \tag{PC*}$$

which is equivalent to (PC). Thus, we have constructed an algorithm to solve (PC) through BDOA and the proximal point method.

D. Our algorithm

In the previous two subsections, we succeed to construct two optimization algorithms for (PB) and (PC). However, our goal in this report is ”develop an algorithm to solve (P0)”. We still need to show how to utilize the algorithms of (PB) and (PC) to generate an algorithm of (P0).

Notice that (P0*) can be expressed as follows:

$$\begin{cases} \arg\min_{x \in \mathbb{R}^n} C_0(x) & + \sum_{i \in \mathcal{H}} k_0 \cdot \mathrm{ReLU}(C_i(x)) \\ & + \sum_{i \in \mathcal{B}} k_0 \cdot \mathrm{ReLU}(C_i(x)). \end{cases}$$

It is exactly an optimization problem to find an optimal point of “the optimized function of (PB*) + the optimized function of (PC*)”. Hence, we can apply the **proximal gradient method** [11] immediately and obtain an algorithm for (P0*). In short, we can express our algorithm in this report as the **Algorithm** 2.

Algorithm 2: Our algorithm

```
Given an initial point $x^0 \in \mathbb{R}^n$, $N_{\text{filter}} > 0$, $k_c > 0$,
  $\alpha_2 > 0$ and iteration number $I$;
for $k := 0$ to $I$ do
    $p^k \leftarrow$ the solution of (P1) at $x = x^k$;
    $\mathcal{H}' := \{ i | \| \langle \nabla^2 C_i(x^k) p^k, p^k \rangle \| \le N_{\text{filter}}, \}$;
    $\mathcal{B}' := \{ i | \| \langle \nabla^2 C_i(x^k) p^k, p^k \rangle \| > N_{\text{filter}}, \}$;
    if $k$ is even then
        $p^k \leftarrow$ the solution of (P1') at $x = x^k$;
        $\alpha^k \leftarrow \alpha_2$;
        // $\alpha_2$ is some positve constant
           here.
        $x^{k+1} \leftarrow x^k + \alpha^k \cdot p^k$;
    else
        $x^{k+1} \leftarrow$ the solution of (P1”) at $x = x^k$;
        // This step is solved through
           proximal point method.
    end
    $k \leftarrow k + 1$;
end
return $x^k$
```

Algorithm 2 is our proposed method for non-linear constrained optimization. We expect our method will be more efficient than the liearnization method. In the next section, we will perform a series of experiments to demonstrate the advantage of our proposed method compared to linearization method.

IV. Experiment

We set the hyper-parameters as follow: $N_{\text{Filter}} = 10$, $\alpha_2 = 1$, $k_c = 50$. The experiment has been conducted with three settings as Table I shown. In setting 1, only exponential constraints are considered. Setting 2 adds a convex polynomial constraint. And in setting 3, only polynomial constraints are considered. In each setting, we respectively use the linearization method and our proposed method to optimize the objective function under the constraints. The number of iterations is set as 20. We introduce a loss function to evaluate the optimization process:

$$\text{Loss}(X) = f_0(X) + \sum_{i\in[m]} k_0 \cdot \text{ReLU}(C_i(X))$$

where f_0 is the objective function, $[m]$ is the index set of constraints, and k_0 - the "penalty" coefficient - is a hyper-parameter greater than zero. Here we set k_0 as 10000. Then we plot the loss values during the optimization process, as shown in Fig. 1, Fig. 2 and Fig. 3. Note that the values are the results of taken the logarithm.

We can see that, in general, our method can converge faster than the linearization method. There exists some cases where the linearization method cannot produce a desirable result, but our method can reach a quite low loss value in comparison.

The implementation details can be found by the link: https://github.com/slimon110/2022_Fault_Tolerance

TABLE I
EXPERIMENT SETTINGS

Setting	Objective Function	Constraints	Initial Point (X_0)
1	$f(x,y) = y^2 - 3$	$10^x - 10^{-1} \leqslant 0$ $10^y - 10^{-3} \leqslant 0$	$(3.0, -2.0)$ $(-4.0, 9.0)$
2	$f(x,y) = y^2 - 3$	$10^x - 10^{-1} \leqslant 0$ $10^y - 10^{-3} \leqslant 0$ $x^2 + y^2 - 20 \leqslant 0$	$(3.0, -2.0)$ $(3.0, -20.0)$
3	$f(x,y) = x^2 + y^2 + 10$	$x^6 + y^6 - 20 \leqslant 0$ $x^2 + y^2 - 10 \leqslant 0$ $y^8 - 20 \leqslant 0$ $x + y - 10 \leqslant 0$ $3x - 2y + 1 \leqslant 0$	$(30.0, -8.0)$ $(-100.0, 5.0)$

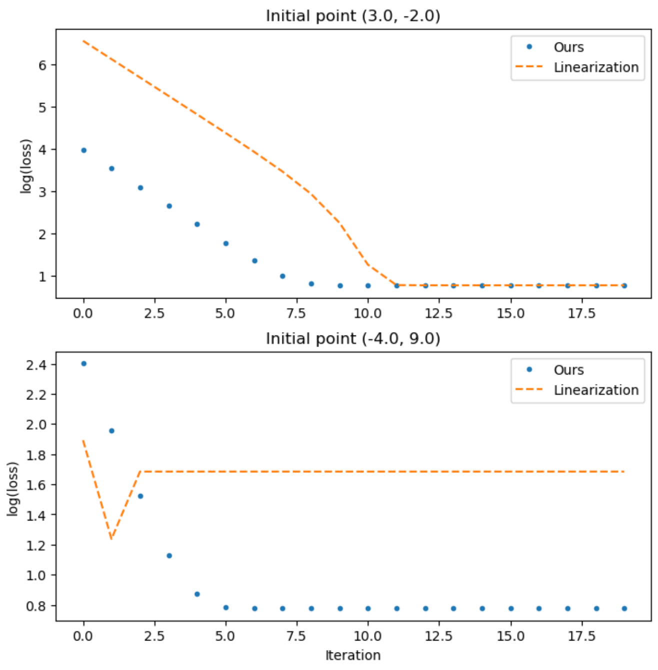

Fig. 1. Experiment result of setting 1.

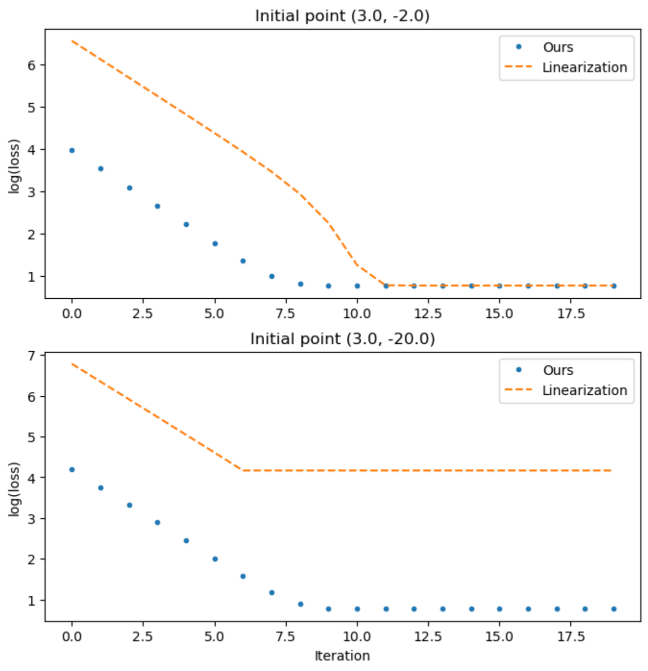

Fig. 2. Experiment result of setting 2.

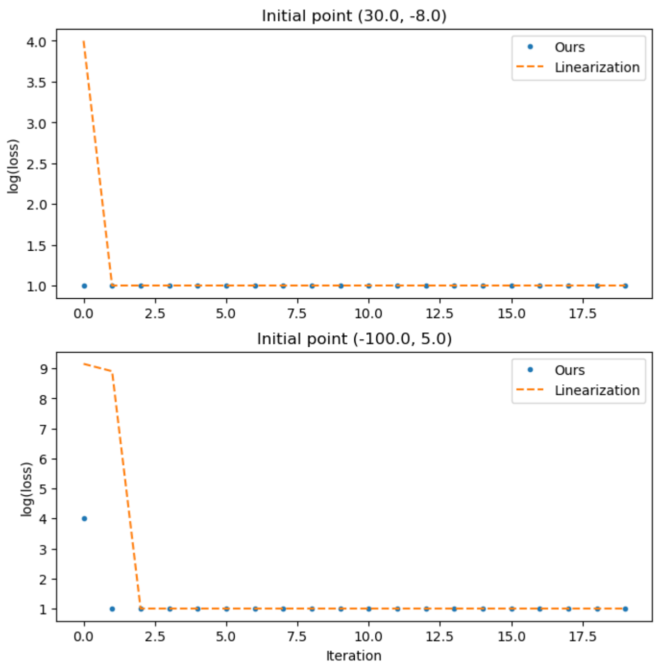

Fig. 3. Experiment result of setting 3.

V. Conclusion

In this report, we construct **Algorithm 2** based on **Byzantine distributed optimization algorithm (BDOA)** [7] and **Linearization** [1] for non-linear constraints optimization problem (P0). The novelty of our algorithm is that we extend the definition of "fault" in Byzantine distributed problem (1) and utilize BDOA as a filter. With the help of BDOA, in our algorithm, we can divide the constrained optimization problem (P0) into two parts, suitable for linear approximation or not. For the part suitable for linear approximation, we optimize it with linearization method; For the part unsuitable for linear approximation, we optimize it with **proximal point method** [10]. Finally, we hybrid the above two parts with **proximal gradient method** [11]. We expect our algorithm would be more efficient than linearization method for non-linear constrained optimization problem (P0), since our algorithm can avoid the step size of liniearization method being too small due to unsuitable linear approximation.

Last but not least, we also perform a series of experiments to demonstrate the advantage of our algorithm. The experiments can be divided into three settings, exponential constraints, polynomial constraints, and "exponential + polynomial constraints" constraints. In all experiments, our algorithm shows a more efficient than linearization method.

In conclusion, in this report, we proposed **Algorithm 2** which performs experimentally better compared to linearization method.

VI. Future works and unsolved problems

There are still some topics that we could research in the future. The first one is the complexity of our algorithm. We have not given a theoretical proof to show our algorithm (**Algorithm 2**) is better than linearization method. The obstacle for us to provide rigorous proof is that it is very hard to relate the complexity with " unsuitable for linear approximation". We hope to prove that the complexity of our algorithm will be less than the complexity of linearization method, when the optimization problem which we consider is sufficiently unsuitable for linear approximation. However, we have no idea how to achieve our goal so far. The second part is that we have not provided an estimation of hyper-parameters. In our algorithm, there are three hyper-parameters, N_{Filter}, α_2, k_c. Sometimes, we need to spend some time on tuning these hyper-parameters in order to obtain better performance. To save time and make our algorithm more efficient for people, we still need to give an estimation of hyper-parameters. The third part is that we want to do more experiments to demonstrate the feasibility of our algorithm. There are so many kinds of constraints we are interested in, but we do not have enough time to do experiments. Following is the list of future works mentioned above:

- Provide theoretical proof to show our algorithm is better than linearization method, when the optimization problem which we consider is sufficiently unsuitable for linear approximation.
- Provide an estimation of hyper-parameters.
- Do more experiments to show the feasibility of our algorithm.

References

[1] S. Wilson and B. Pshenichnyj, *The Linearization Method for Constrained Optimization*. Springer Series in Computational Mathematics, Springer Berlin Heidelberg, 2012.

[2] W. T. Li, X. W. Shi, Y. Q. Hei, S. F. Liu, and J. Zhu, "A hybrid optimization algorithm and its application for conformal array pattern synthesis," *IEEE Transactions on antennas and propagation*, vol. 58, no. 10, pp. 3401–3406, 2010.

[3] C. Xu, Q. Liu, and T. Huang, "Resilient penalty function method for distributed constrained optimization under byzantine attack," *Information Sciences*, vol. 596, pp. 362–379, 2022.

[4] L. Su and N. H. Vaidya, "Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms," in *Proceedings of the 2016 ACM symposium on principles of distributed computing*, pp. 425–434, 2016.

[5] N. Gupta and N. H. Vaidya, "Byzantine fault tolerant distributed linear regression," *arXiv preprint arXiv:1903.08752*, 2019.

[6] N. Gupta and N. H. Vaidya, "Byzantine fault-tolerant parallelized stochastic gradient descent for linear regression," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 415–420, IEEE, 2019.

[7] S. Liu, N. Gupta, and N. H. Vaidya, "Approximate byzantine fault-tolerance in distributed optimization," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pp. 379–389, 2021.

[8] N. Gupta, S. Liu, and N. H. Vaidya, "Byzantine fault-tolerant distributed machine learning using stochastic gradient descent (sgd) and norm-based comparative gradient elimination (cge)," *arXiv preprint arXiv:2008.04699*, 2020.

[9] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–25, 2017.

[10] "Proximal point method." http://www.seas.ucla.edu/~vandenbe/236C/lectures/ppm.pdf.

[11] "Proximal gradient method." https://www.stat.cmu.edu/~ryantibs/convexopt/lectures/prox-grad.pdf.