Asynchronous Non-Convex Optimization for Separable Problems

Sandeep Kumar SANDKR@IITK.AC.IN
Ketan Rajawat KETAN@IITK.AC.IN
Dept. Of. EE, IIT Kanpur
Kanpur-208016, India

Abstract

This paper considers the distributed optimization of a sum of locally observable, non-convex functions. The optimization is performed over a multi-agent networked system, and each local function depends only on a subset of the variables. An asynchronous and distributed alternating directions method of multipliers (ADMM) method that allows the nodes to defer or skip the computation and transmission of updates is proposed. The algorithm can tolerate any bounded level of asynchrony and converges to local minima under certain regularity conditions.

1. Introduction

Multi-agent networked systems arise in a number of engineering disciplines such as tactical ad hoc networks, environmental monitoring networks, multi-robot control and tracking, internet-scale monitoring, and large-scale learning. The estimation, resource allocation, and network control tasks required in these applications are often formulated as distributed optimization problems, where each node is associated with a local cost function, determined from the set of local and possibly private measurements (Boyd et al., 2011). The nodes must nevertheless cooperate in order to minimize the network objective function, which is the sum of local costs. Practical networks are also heterogeneous with respect to their processing powers, energy availability, and communication capabilities, giving rise to asynchrony (Bertsekas & Tsitsiklis, 1989). Indeed, a key feature required of the distributed algorithms is their tolerance to processing and communication delays.

In general, distributed optimization algorithms are designed either in the primal (Bertsekas & Tsitsiklis, 1989; Duchi et al., 2012), or dual domain (Boyd et al., 2011; Hong, 2014). A popular dual approach is the distributed alternating direction method of multipliers (ADMM), where the nodal variables are decoupled through the introduction of the consensus constraints, and the updates are carried out in the dual domain (Boyd et al., 2011). High-dimensional problems are handled through the so-called *general-form consensus* formulation, where local updates depend only on a subset of optimization variables(Boyd et al., 2011). The ADMM algorithm is also applicable to a class of non-convex problems, where it has been shown to converge to a local minimum (Hong, 2014), but are limited to star topology with a dedicated fusion center. The algorithm design becomes challenging in the absence of a centralized controller or a fusion center, where nodal interactions are limited to their neighborhoods, and global network state information is largely unavailable (Kumar et al., 2016; Boyd et al., 2011). In high-dimensional problems, even local message passing may be prohibitive, since each node may only be interested in a subset of the optimization variables.

This work considers the non-convex general-form consensus optimization problem arising in a multi-agent networked system. The first contribution is the development of an asynchronous and distributed ADMM framework that runs without a fusion center. The algorithm allow the nodes to, at times, skip the computationally intensive steps and/or the transmission of updates. As the second contribution, it is shown that it converges to a local minimum, under certain regularity conditions. The convergence analysis reveals that with appropriately chosen parameters, the algorithms can tolerate any bounded level of asynchrony.

2. Problem Formulation

This section details the partially separable non-convex problem formulation. Consider a network represented by the undirected graph $\mathcal{G} = (\mathcal{K}, \mathcal{E})$, where $\mathcal{K} := \{1, 2, \ldots, K\}$ denotes the set of agents or nodes and $\mathcal{E}$, the set of edges that represent communication links. A node $k \in \mathcal{K}$ may only communicate with its neighbors $\mathcal{N}'_k := \{j | (j,k) \in \mathcal{E}\}$. Consider first the general multi-

Appearing in Proceedings of the 2^{nd} Indian Workshop on Machine Learning, IIT Kanpur, India, 2016. Copyright 2016 by the author(s).

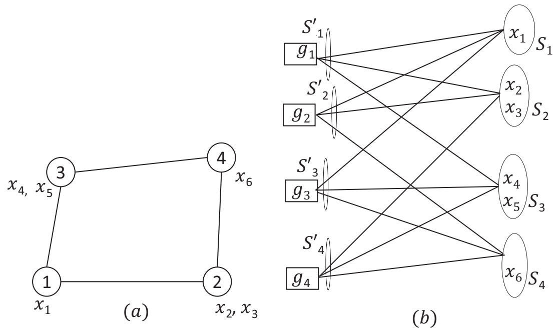

Figure 1. (a)An example of a four-node network. (b)Factor graph representation for the objective function of (2)

agent problem where the nodes want to cooperatively solve the following optimization problem

$$\min f(\mathbf{x}) := \sum_{k=1}^{K} g_k(\mathbf{x}) + h(\mathbf{x}) \quad \text{s. t. } \mathbf{x} \in \mathcal{X} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^{N\times 1}$, $g_k : \mathbb{R}^N \to \mathbb{R}$ for $k = 1, \ldots, K$ are differentiable, possibly non-convex functions, and $h : \mathbb{R}^N \to \mathbb{R}$ is a convex but not necessarily differentiable function. The set $\mathcal{X}$ is closed, convex, and compact. Within the distributed setting considered here, the function $g_k(\mathbf{x})$ is local to node k, and the network has no central coordinator or fusion center.

In general, non-convex problems such as 1 are solved in a distributed manner using the first order gradient or subgradient descent, dual methods such as ADMM, convex relaxation (such as semidefinite relaxation), successive convex optimization, or leveraging the problem structure. These approaches result in algorithms that are parallelizable to various extents, with different computational and message passing requirements. Of particular interest here is the high-dimensional regime, where large N prohibits nodes from operating over and exchanging the full vector $\mathbf{x}$. To this end, the next section considers a partially separable form of 1 which is amenable to a distributed optimization algorithm.

2.1. Partially separable form

Consider a special case of (1) where the optimization variables $\{x_n\}_{n=1}^N$ are also partitioned among nodes, and each variable is of interest to exactly one node. To this end, let $\{\mathcal{S}_k\}_{k=1}^K$ denote disjoint subsets such that the variables $\{x_n | x_n \in S_k\}$ are local to node k. Further, the component function $g_k(\cdot)$ at node k depends only on variables that are local either to node k or its neighbors. The overall problem considered here takes the following form:

$$P = \min_{\mathbf{x}} \sum_{k=1}^{K} g_k(\{x_n\}_{n\in\mathcal{S}'_k}) + h_k(\{x_n\}_{n\in\mathcal{S}_k}) \tag{2}$$
$$\text{s. t. } \{x_n\}_{n\in\mathcal{S}_k} \in \mathcal{X}_k \quad k = 1, 2, \ldots, K.$$

where the set $\mathcal{S}'_k := \bigcup_{j\in\{k\}\cup\mathcal{N}'_k} \mathcal{S}_j$. Observe here that for each node k, the function h_k and the constraint set $\mathcal{X}_k$ depend only on the variables in $\mathcal{S}_k$. For this partially separable form, it is now possible to express the objective function as a bipartite factor graph, with check nodes representing the summands g_k, and the variable nodes representing the sets $\mathcal{S}_k$. For the four-node example network shown in Fig. 1(a), the factor graph is shown in Fig. 1(b). From the perspective of algorithm design, the dependence structure imposed by (2) can be exploited to eliminate message passing between non-neighboring nodes. The partially separable form considered here occurs commonly in the context of distributed estimation. But, cooperation between nodes is still required here, since parameters at neighboring nodes are often coupled or correlated.

3. Distributed Asynchronous ADMM

This section details the proposed distributed asynchronous algorithm for solving (2) via ADMM, and provides the relevant convergence results. For ease of exposition, it is assumed henceforth that the sets $\mathcal{S}_k$ are singletons, i.e., $\mathcal{S}_k = \{k\}$. Note however that the algorithms and the corresponding convergence results are applicable to the general case as well. Towards this end, introduce copies x_{kj} of the variable x_j corresponding to all nodes $j \in \mathcal{N}_k$ for all $k \in \mathcal{K}$. To make sure that the copies of the a variable agree across nodes, also introduce the consensus variable z_k for each neighborhood $\mathcal{N}_k$. The introduction of these extra variables amounts to reformulating (2) as

$$\min_{\{\mathbf{x}_k\},\mathbf{z}} \sum_{k=1}^{K} g_k(\{x_{kj}\}_{j\in\mathcal{N}_k}) + h_k(z_k) \tag{3}$$
$$\text{s. t.} \quad x_{kj} = z_j, \quad j \in \mathcal{N}_k, \quad k = 1, \ldots, K \tag{4}$$
$$z_k \in \mathcal{X}_k, \qquad k = 1, \ldots, K$$

where $\mathbf{x}_k$ collects the variables required at node k, i.e., $[\mathbf{x}_k]_j = x_{kj}$ for $j \in \mathcal{N}_k$, and zero otherwise. The idea of introducing consensus variables, in order to make the updates separable in the optimization variables, is well known (Boyd et al., 2011). It is now possible to apply the ADMM method by associating dual variables y_{kj} for each constraint in (4) and writing the augmented Lagrangian as

$$L(\{\mathbf{x}_k\}, \mathbf{z}, \{\mathbf{y}_k\} = \sum_{k=1}^{K} \Bigg(g_k(\mathbf{x}_k) + h_k(z_k)$$
$$+ \Big(\sum_{j\in\mathcal{N}_k} \langle y_{kj}, x_{kj} - z_j \rangle + \frac{\rho_k}{2} \|x_{kj} - z_j\|^2 \Big) \Bigg) \tag{5}$$

where $\rho_k > 0$ is a positive penalty parameter and $\mathbf{z}$ collects $\{\mathbf{z}_k\}_{k\in\mathcal{K}}$. On the left-hand side, for each k, the dual vector $\mathbf{y}_k \in \mathbb{R}^N$ is such that $[\mathbf{y}_k]_j = y_{kj}$ if $j \in \mathcal{N}_k$, and zero otherwise. In particular, starting with arbitrary $\{x^1_{kj}\}$ and $\{y^1_{kj}\}$, the update for $\{z^{t+1}_j\}$ are evaluated as

$$z^{t+1}_j = \text{prox}_j\left(\frac{\sum_{k\in\mathcal{N}_j}\left(\rho_k x^t_{kj} + y^t_{kj}\right)}{\sum_{k\in\mathcal{N}_j}\rho_k}\right) \tag{6}$$

where the proximal point function $\text{prox}_j(\cdot)$ is defined as $\text{prox}_j := \arg\min_{u\in\mathcal{X}_j} h(u) + \frac{1}{2}\|x-u\|^2$. Next for the $\mathbf{x}_k$ update, observe that since the component functions $g_k(\cdot)$ are non-convex, the exact update is difficult to carry out. By linearizing the function $g_k(\mathbf{x}_k)$ at $\mathbf{z}^{t+1}_k$, the update $\mathbf{x}^{t+1}_k$ can however be calculated approximately as follows

$$\mathbf{x}^{t+1}_k \approx \arg\min_{\mathbf{x}_k} g_k(\mathbf{z}^{t+1}_k) + \langle\nabla g_k(\mathbf{z}^{t+1}_k), \mathbf{x}_k - \mathbf{z}^{t+1}_k\rangle + \sum_{j\in\mathcal{N}_k}\langle y^t_{kj}, x_{kj} - z^{t+1}_j\rangle + \sum_{j\in\mathcal{N}_k}\frac{\rho_k}{2}\left\|x_{kj} - z^{t+1}_j\right\|^2$$

where the vector $[\mathbf{z}_k]_j := z_j$ for all $j \in \mathcal{N}_k$ and zero otherwise. Since nodal functions $g_k(\cdot)$ depend only on $\{x_n\}_{n\in\mathcal{N}_k}$, the gradient vector is defined as

$$[\nabla g_k(\mathbf{z}^{t+1}_k)]_j := \begin{cases} \frac{\partial}{\partial x_{kj}} g_k(\mathbf{x}_k)\Big|_{\mathbf{x}_k=\mathbf{z}_k} & j \in \mathcal{N}_k \\ 0 & j \notin \mathcal{N}_k. \end{cases} \tag{7}$$

The approximate update of x^{t+1}_{kj} thus becomes

$$x^{t+1}_{kj} = z^{t+1}_j - \frac{1}{\rho_k}\left([\nabla g_k(\mathbf{z}^{t+1}_k)]_j + y^t_{kj}\right), j \in \mathcal{N}_k \tag{8}$$

Finally, dual updates for $1 \le k \le K$ are given by

$$y^{t+1}_{kj} = y^t_{kj} + \rho_k\{x^{t+1}_{kj} - z^{t+1}_j\}, \qquad j \in \mathcal{N}_k. \tag{9}$$

The classical proximal ADMM presented here is synchronous and is therefore cannot be readily implemented in a heterogeneous network. For instance, the computationally intensive steps, namely, calculation of $\nabla g_k(\cdot)$ in (8) or $\text{prox}_k(\cdot)$ in (6) are required to be carried out by all nodes at every iteration. In other words, the network must wait for the slowest node to carry out its update.

Towards addressing these challenges, we introduce an algorithm that allow the updates to be skipped or carried out with an old copy of the gradient vector. Specifically, it is allowed that the calculation of $\nabla g_k(\cdot)$ or $\text{prox}_k(\cdot)$ be skipped for some iterations. For each non-updating node this is achieved by simply setting $z^{t+1}_j = z^t_j$ $j \in \mathcal{K}$ at time t. Define $\mathcal{S}^t$ as the set of nodes that carry out the update at time t, then the z^{t+1}_j update can be written as

$$z^{t+1}_j = \begin{cases} \text{prox}_j\left(\frac{\sum_{k\in\mathcal{N}_j}\left(\rho_k x^t_{kj} + y^t_{kj}\right)}{\sum_{k\in\mathcal{N}_j}\rho_k}\right) & j \in \mathcal{S}^t \\ z^t_j & j \notin \mathcal{S}^t \end{cases} \tag{10}$$

Whenever $j \notin \mathcal{S}^t$, the subsequent transmission may not be carried out either, and the non-updating node may simply stay silent. The neighboring nodes will then wait for a fixed amount of time to receive an update, and assume that $z^{t+1}_j = z^t_j$ holds for all nodes $j \in \mathcal{N}_k$ that do not transmit anything.

Secondly, the update of $\mathbf{x}^{t+1}_k$ can be carried, by using the latest available gradient $\nabla g_k(\mathbf{z}^{[t+1]_k}_k)$ only, where $t+1-T_k \le [t+1]_k \le t+1$ for some $T_k < \infty$. In other words, the gradient calculation may only be carried out intermittently, and the same gradient can be used for next several time slots. Alternatively, for computationally challenged nodes, the gradient calculation itself may take several time slots. Dropping the subscript k from $[t+1]$ for notational brevity, the asynchronous update of $\mathbf{x}^{t+1}_k$ becomes

$$x^{t+1}_{kj} = z^{t+1}_j - \frac{1}{\rho_k}\left([\nabla g_k(\mathbf{z}^{[t+1]}_k)]_j + y^t_{kj}\right) j \in \mathcal{N}_k \tag{11}$$

Algorithm 1 summarizes the implementation of distributed asynchronous ADMM for networked applications.

Algorithm 1 Distributed Asynchronous ADMM with Optional Updates

```
 1: Set $t = 1$, initialize $\{x^1_{kj}, y^1_{kj}, z^1_j\}$ for all $j \in \mathcal{N}_k$.
 2: for $t = 1, 2, \ldots$ do
 3:     (Optional) Send $\{\rho_k x^t_{kj} + y^t_{kj}\}$ to neighbors $j \in \mathcal{N}_k$
 4:     if $\{\rho_j x^t_{jk} + y^t_{jk}\}$ received from all $j \in \mathcal{N}_k$ then
 5:         (Optional) Update $z^{t+1}_k$ as in (10) and transmit to each $j \in \mathcal{N}_k$
 6:     end if
 7:     if $z^{t+1}_j$ not received from some $j \in \mathcal{N}_k$ then
 8:         set $z^{t+1}_j = z^t_j$
 9:     end if
10:     (Optional) Calculate gradient $\nabla g_k(\mathbf{z}^{t+1}_k)$
11:     Update the primal variable $\mathbf{x}^{t+1}_k$ as in (11)
12:     Update the dual variable $y^{t+1}_{kj}$ as in (9)
13:     if $\|\mathbf{x}^{t+1}_k - \mathbf{x}^t_k\| \le \delta$ then
14:         terminate loop
15:     end if
16: end for
```

3.1. Convergence Analysis for Algorithm 1

In particular, it is shown that the algorithm converges as long as the optional updates happen "often enough." Specifically, recall that $t+1-[t+1] \le T_k$, which implies that the gradients may be calculated using updates that are at most T_k-old. Similarly, let the frequency of update in (10) being carried out at node k be denoted by $0 < f_k \leq 1$. For instance, if the update occurs once every K time slots, $f_k = 1/K$. Then the following assumptions are required.

Assumption A1. *For each node k, the component function gradient $\nabla g_k(\mathbf{x})$ is Lipschitz continuous, that is, there exists $L_k > 0$, for all $\mathbf{x}, \mathbf{x}' \in \text{dom} g_k$ such that*

$$\|\nabla g_k(\mathbf{x}) - \nabla g_k(\mathbf{x}')\| \leq L_k \|\mathbf{x} - \mathbf{x}'\|. \tag{12}$$

Assumption A2. *The set $\mathcal{X}$ is a closed, convex, and compact. The functions $g_k(x)$ is bounded from below over $\mathcal{X}$.*

Assumption A3. *For node k, the step size ρ_k is chosen large enough such that, it holds that $\alpha_k > 0$ and $\beta_k > 0$, where*

$$\alpha_k := \frac{\rho_k f_k}{2} - \left(\frac{7L_k}{2\rho_k^2} + \frac{1}{\rho_k}\right) |\mathcal{N}_k| L_k^2 (T_k+1)^2 - \frac{|\mathcal{N}_k| L_k T_k^2}{2}$$

$$\beta_k := \rho_k - 7L_k \tag{13}$$

Of these, Assumptions (A1) and (A2) are standard in the context of non-convex optimization (Hong, 2014; Boyd et al., 2011) and are satisfied for most problems of interest. Finally, Assumption (A3) specifies the exact relationship that the algorithm and problem parameters $\{L_k, \rho_k, f_k, T_k\}_{k=1}^K$ must satisfy in the worst case.

Lemma 1. *(a) Starting from any time $t = t_0$, there exists $T < \infty$ such that*

$$\begin{aligned} &L(\{\mathbf{x}_k^{T+t_0}\}; \mathbf{z}^{T+t_0}, \{\mathbf{y}_k^{T+t_0}\}) - L(\{\mathbf{x}_k^{t_0}\}; \mathbf{z}^{t_0}, \{\mathbf{y}_k^{t_0}\}) \\ &\leq -\sum_{i=t_0}^{T+t_0-1} \sum_{k=1}^{K} \frac{\beta_k}{2} \sum_{j\in\mathcal{N}_k} \|x_{kj}^{i+1} - x_{kj}^i\|^2 \\ &\qquad - \sum_{i=t_0}^{T+t_0} \sum_{k=1}^{K} \alpha_k \sum_{j\in\mathcal{N}_k} \|z_j^{i+1} - z_j^i\|^2. \end{aligned} \tag{14}$$

(b) The augmented Lagrangian values in (5) are bounded from below, i.e., for any time $t \geq 1$, it holds that Lagrangian satisfies

$$L(\{\mathbf{x}_k^t\}; \mathbf{z}^t, \{\mathbf{y}_k^t\}) \geq \mathsf{P} - \frac{L_k}{2} \sum_{j\in\mathcal{N}_k} \text{diam}^2(\mathcal{X}) > -\infty$$

Lemma 1(a) establishes that there exists some finite T such that the augmented Lagrangian values are non-increasing after T iterations. In practice, the value of T depends on the update frequencies $\{f_k\}_{k=1}^K$. For instance, T could be the minimum number of iterations in which each node k updates z_k^t at least $f_k T$ times. Lemma 1(b) establishes that the Lagrangian is bounded from below.

Theorem 1. *(a) The iterates generated by Algorithm 1 converges in the following sense*

$$\lim_{t\to\infty} \left\|z_k^{t+1} - z_k^t\right\| = 0, \quad \forall \quad k \tag{15a}$$

$$\lim_{t\to\infty} \left\|x_{kj}^{t+1} - x_{kj}^t\right\| = 0, \quad j \in \mathcal{N}_k, \quad \forall \quad k \tag{15b}$$

$$\lim_{t\to\infty} \left\|y_{kj}^{t+1} - y_{kj}^t\right\| = 0, \quad j \in \mathcal{N}_k, \quad \forall \quad k \tag{15c}$$

(b) For each $k \in \mathcal{K}$ and $j \in \mathcal{N}_k$, denote limit points of the sequences $\{z_k^t\}$, $\{x_{kj}^t\}$, and $\{y_{kj}^t\}$ by $z_k^\star$, $x_{kj}^\star$, and $y_{kj}^\star$, respectively. Then $\{\{z_k^\star\}, \{x_{kj}^\star\}, \{y_{kj}^\star\}\}$ is a stationary point of (3) and satisfies

$$\nabla g_k(\mathbf{x}_k^\star) + \mathbf{y}_k^\star = 0, \quad k = 1, \ldots, K \tag{16a}$$

$$\sum_{j\in\mathcal{N}_k} y_{jk}^\star \in \partial(h_k(z)) \mid_{z=z_k^\star} \quad k = 1, \ldots, K \tag{16b}$$

$$x_{kj}^\star = z_j^\star \in \mathcal{X}_j, \quad j \in \mathcal{N}_k, k = 1, \ldots, K \tag{16c}$$

Note that it suffices to show that Algorithm 1 converges to a stationary solution of (3) which is equivalent to (2). In other words, $\{z_k^\star\}_{k=1}^K$ can be used as a solution to (2). It is emphasized that Algorithm 1 may not necessarily converge to a globally optimum solution to (2). The proof of the theorem with detailed analysis of the non-convex ADMM framework is deferred to the longer version (Kumar et al., 2016).

Dual submission

The detailed manuscript of this work, is under review for the publication in, *IEEE Transactions on Signal and Information Processing over Networks.*

References

Bertsekas, Dimitri P and Tsitsiklis, John N. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

Boyd, Stephen, Parikh, Neal, Chu, Eric, Peleato, Borja, and Eckstein, Jonathan. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trend. Mach. Learn.*, 3(1):1–122, 2011.

Duchi, John C, Agarwal, Alekh, and Wainwright, Martin J. Dual averaging for distributed optimization: convergence analysis and network scaling. *IEEE Trans. on Automatic control,*, 57(3):592–606, 2012.

Hong, Mingyi. A distributed, asynchronous and incremental algorithm for nonconvex optimization: An ADMM based approach. *arXiv preprint arXiv:1412.6058*, 2014.

Kumar, Sandeep, Jain, Rahul, and Rajawat, Ketan. Asynchronous optimization over heterogeneous networks via consensus admm. *arXiv preprint*, 2016. URL `http://arxiv.org/pdf/1605.00076v1.pdf`.