Distributed Constrained Nonconvex Optimization: the Asynchronous Method of Multipliers

Francesco Farina\textsuperscript{1}, Andrea Garulli\textsuperscript{1}, Antonio Giannitrapani\textsuperscript{1}, and Giuseppe Notarstefano\textsuperscript{2}

\textsuperscript{1}Dipartimento di Ingegneria dell’Informazione e Scienze Matematiche, Università di Siena, Siena, Italy.
\textsuperscript{2}Department of Engineering, Università del Salento, Lecce, Italy.

Abstract

This paper presents a fully asynchronous and distributed approach for tackling optimization problems in which both the objective function and the constraints may be nonconvex. In the considered network setting each node is active upon triggering of a local timer and has access only to a portion of the objective function and to a subset of the constraints. In the proposed technique, based on the method of multipliers, each node performs, when it wakes up, either a descent steps on a local augmented Lagrangian or an ascent step on the local multiplier vector. Nodes realize when to switch from the descent step to the ascent one through an asynchronous distributed logic-AND, which detects when all the nodes have reached a predefined tolerance in the minimization of the augmented Lagrangian. It is shown that the resulting distributed algorithm is equivalent to a block coordinate descent for the minimization of the global non-separable augmented Lagrangian. This allows one to extend the properties of the centralized method of multipliers to the considered distributed framework. Two application examples are presented to validate the proposed approach: a distributed source localization problem and the parameter estimation of a neural classifier.

1 Introduction

Nonconvex optimization problems are commonly encountered when dealing with control, estimation and learning within cyber-physical networks. In these contexts, typically each device knows only a portion of the whole objective function and a subset of the constraints, so that, to avoid the presence of a central coordinator, distributed algorithms are needed. Distributed optimization methods relevant to the problem considered in this paper can be divided in two groups: those handling nonconvex functions but not local constraints, and methods handling local constraints, designed for convex problems.

Distributed algorithms for nonconvex optimization have started to appear in the literature only recently. Regarding constrained problems, most of the proposed methods usually
do not deal with local constraints. In [1] a stochastic gradient method is proposed to minimize the sum of nonconvex functions subject to a constraint known to all agents. In [2] a decentralized Frank-Wolfe method for finding a stationary point of the sum of differentiable and nonconvex functions is given. In [3, 4] the authors propose distributed algorithms, respectively for balanced and general directed graphs, based on the idea of tracking the whole function gradient and performing successive convex approximation of the nonconvex cost function. A perturbed push-sum algorithm for the unconstrained minimization of the sum of nonconvex functions is given in [5].

Distributed optimization algorithms handling local constraints are basically designed for convex problems, except for some specific problem settings. In [6], the authors propose a distributed random projection algorithm, while a proximal based algorithm is presented in [7]. A subgradient projection algorithm taking into account communication delays is given in [8]. In [9] randomized block-coordinate descent methods are employed, to solve convex optimization problems with linearly coupled constraints over networks. As for Lagrangian based distributed algorithms, an edge-based method for convex optimization problems is proposed in [10]. An iterative scheme combining dual decomposition and proximal minimization is presented in [11]. In [12] an asynchronous distributed alternating directions method of multipliers (ADMM) is proposed for a separable, constrained optimization problem. Other ADMM based approaches are presented in [13, 14, 15], while an asynchronous proximal dual algorithm is proposed in [16]. A distributed algorithm for a structured class of nonconvex optimization problems with local constrains is presented in [17].

The main contribution of this paper is a fully distributed optimization algorithm addressing constrained optimization problems over networks, in which both local cost functions and local constraints may be nonconvex. The proposed algorithm, hereafter referred to as ASynchronous Method of Multipliers (ASYMM), works in asynchronous networks in which nodes are typically in idle mode and wake up when a local timer triggers. ASYMM features two types of local updates at each node, a primal and a multiplier one, which are regulated by an asynchronous distributed logic-AND algorithm. When awake, nodes start performing a descent step on a local augmented Lagrangian until a given local tolerance is reached. Once a node has reached its own tolerance, it starts spreading this information over the network. By means of the distributed logic-AND algorithm, the nodes realize when all of them have reached the tolerance and thus, still asynchronously, they start performing the multiplier updates. An interesting feature of ASYMM is that a node does not need to wait for all multiplier updates to start again its primal descent, but rather it just needs to receive all neighboring multipliers. The analysis of ASYMM is based on showing that it implements a suitable inexact version of the Method of Multipliers, in which the primal minimization is performed by means of a block coordinate descent algorithm up to a given tolerance. Thanks to this connection, the ASYMM distributed algorithm inherits the main properties of the centralized method [18, 19]. Indeed, by adapting the results in [20], it is shown that the block coordinate minimization is guaranteed to converge to a stationary point of the augmented Lagrangian, up to a given tolerance.

Under a further technical condition, exploiting the fact that the augmented Lagrangian tends to become (locally) strongly convex as the penalty parameters increase, a bound on the whole augmented Lagrangian gradient, based on the local tolerances, is provided. Thus, if the local tolerances converge to zero, ASYMM converges to a local minimum of the nonconvex constrained optimization problem. It is worth mentioning that our algorithm implicitly leverages on some results given in [21, 22] to handle non-regularity of local minima (due to the presence of local copies of the common decision variable). With respect to the distributed algorithm proposed in [21, 22], the main novelty of ASYMM is that the network
model is asynchronous and switching from a primal to a multiplier update is performed by nodes in a distributed way.

Finally, it is shown that the proposed algorithm can effectively solve also big-data problem (i.e., with a high dimensional decision variable). Indeed, thanks to its block-wise structure, each agent can optimize over and transmit only one block of the entire solution estimate.

The paper is organized as follows. In Section 2, the distributed optimization set-up is presented. The proposed algorithm is presented in Section 3 and analyzed and discussed in Section 4. In Section 5, an extension for dealing with high dimensional optimization problems is presented. Finally, two numerical application are presented in Section 6 and some conclusions are drawn in Section 7.

2 Set-up and Preliminaries

2.1 Notation and definitions

Given a matrix $A \in \mathbb{R}^{n \times m}$ we denote by $A[i,j]$ the $(i,j)$-th element of $A$, by $A[i,\cdot]$ its $i$-th column, by $A[\cdot,j]$ its $i$-th row and by $A[i,j,k]$ the elements of the $k$-th column of $A$ from row $i$ to $j$. We write $A[i,\cdot] = b$ to assign the value $b$ to all the elements in the $i$-th row of $A$.

Two vectors $a,b \in \mathbb{R}$ and a constant $c$, we write $a > c$ if all elements of $a$ are greater than $c$ and $a > b$ if $a[i] > b[i]$ for all $i$. If $J = \{j_1,\ldots,j_m\}$ is a set of indexes, we denote by $[x]_{j \in J}$ the vector $[x_{j_1},\ldots,x_{j_m}]^\top$.

The following definitions will be useful in the following.

A function $\Psi(x)$ has Lipschitz continuous gradient if there exists a constant $L$ such that
$$\|\nabla \Psi(x) - \nabla \Psi(y)\| \leq L\|x - y\|$$
for all $x,y$. It is $\sigma$-strongly convex if $(\nabla \Psi(x) - \nabla \Psi(y))^\top(x - y) \geq \sigma\|x - y\|^2$.

Let $x = [x_1^\top,\ldots,x_N^\top]^\top$, with $x_i \in \mathbb{R}^{n_i}$ and $\sum_{i=1}^N n_i = n$, and let $U_{n \times n}$, with $U_i \in \mathbb{R}^{n_i \times n_i}$, be a partition of the identity matrix such that $x = \sum_{i=1}^N U_i x_i$ and $x_i = U_i^\top x$. The function $\Phi(x)$ has block component-wise Lipschitz continuous gradient if there are constants $L_i \geq 0$ such that $\|\nabla_x \Phi(x + U_i s_i) - \nabla_x \Phi(x)\| \leq L_i \|s_i\|$ for all $x \in \mathbb{R}^n$ and $s_i \in \mathbb{R}^{n_i}$.

We say that indexes in $\{1,\ldots,N\}$ are drawn according to an essentially cyclic rule if there exists $M \geq N$ such that every $i \in \{1,\ldots,N\}$ is drawn at least once every $M$ extractions.

2.2 Distributed Optimization Problem

Consider the following optimization problem

$$\min_{x} \sum_{i=1}^N f_i(x) \quad \text{subject to} \quad h_i(x) = 0, \quad i = 1,\ldots,N,$$
$$g_i(x) \leq 0, \quad i = 1,\ldots,N,$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$, $h_i : \mathbb{R}^n \to \mathbb{R}^{m_i}$ and $g_i : \mathbb{R}^n \to \mathbb{R}^{m'_i}$.

Assumption 1. Functions $f_i$ and each component of $h_i, g_i$ are of class $C^2$ and have bounded Hessian.

Assumption 2. Problem (1) has at least one feasible solution and every local minimum of (1) is a regular point and satisfies the second order sufficiency conditions.
The aim of the paper is to present a method for solving problem (1) in a distributed way, by employing a network of $N$ peer processors without a central coordinator. Each processor has a local memory, local computation capability and can exchange information with neighboring nodes. Moreover, functions $f_i$, $h_i$ and $g_i$ are private to node $i$. The network is described by a fixed, undirected and connected graph $G = (V,E)$, where $V = \{1,...,N\}$ is the set of nodes and $E \subseteq \{1,...,N\} \times \{1,...,N\}$ is the set of edges. We denote by $N_i = \{j \in V | (i,j) \in E\}$ the set of the neighbors of node $i$ and by $d_i = |N_i| + 1$. Also, we denote by $d_G$ the diameter of $G$.

Regarding the communication protocol, a generalized version of the asynchronous model presented in [16] is considered. Each node has its own concept of time defined by a local clock $s_i$. As long as $s_i < T_i$ the node is in IDLE mode, which triggers when the node has to awake, independently of the other nodes. Between two triggering events each node is in IDLE mode, i.e., it listens for messages from neighboring nodes and, if needed, updates some local variables, but it does not broadcast any information. When a trigger occurs, it switches into AWAKE mode, performs local computations and sends the updated information to neighbors. Formally, the triggering process is modeled by means of a local clock $s_i \in \mathbb{R}_{>0}$ and a random waiting time $T_i$. As long as $s_i < T_i$ the node is in IDLE. When $s_i = T_i$ the node switches to the AWAKE mode and, after running the local computations, it resets $s_i = 0$ and generates a new waiting time $T_i$.

**Assumption 3** (Local timers). For each node $i$, there exists a constant $\bar{T}_i$ such that $T_i \leq \bar{T}_i$, for all realization of the waiting time $T_i$. □

**Assumption 4** (No simultaneous awakening). Only one node can be awake at each time instant. □

### 2.3 Equivalent Formulation and Method of Multipliers

Because of the connectedness of $G$, problem (1) can be rewritten in the equivalent form

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{N} f_i(x_i) \\
\text{subject to} & \quad x_i = x_j, \quad \forall (i,j) \in E, \\
& \quad h_i(x_i) = 0, \quad \forall i \in V, \\
& \quad g_i(x_i) \leq 0, \quad \forall i \in V,
\end{align*}
\]

where $x_i \in \mathbb{R}^n$ for all $i \in V$.

Let us now introduce the augmented Lagrangian associated to problem (2). Let $\nu_{ij} \in \mathbb{R}^n$ and $\rho_{ij} \in \mathbb{R}$ be the multiplier and penalty parameter associated to the equality constraint $x_i = x_j$. We compactly define $\nu_i = [\nu_{ij}]_{j \in N_i}, \rho_i = [\rho_{ij}]_{j \in N_i}$. Similarly, let $\lambda_i \in \mathbb{R}^{m_i}$ and $\theta_i \in \mathbb{R}$ (respectively $\mu_i \in \mathbb{R}^{m_i'}$ and $\zeta_i \in \mathbb{R}$) be the multiplier and penalty parameter associated to the equality (respectively inequality) constraint of node $i$. Moreover, let $x = [x_1^\top,...,x_N^\top]^\top$; denote by $p = [p_i, \theta_i, \zeta_i]_{i \in V}^\top$ the vector stacking all the penalty parameters; $\nu = [\nu_i]_{i \in V}$, $\lambda = [\lambda_i]_{i \in V}$ and $\mu = [\mu_i]_{i \in V}$ be the vectors stacking the corresponding multipliers, and, consistently, let $\theta = [\nu^\top, \lambda^\top, \mu^\top]^\top$. Let us define the following $g$ operator

\[
g_q(a,b) = \frac{1}{2c} \left( \max\{0,a+cb\}^2 - a^2 \right).
\]

Then, the augmented Lagrangian associated to (2) is defined as

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{N} f_i(x_i) + \frac{\rho}{2} \sum_{i=1}^{N} \sum_{j \neq i} \nu_{ij} (x_i - x_j)^2 \\
\text{subject to} & \quad x_i = x_j, \quad \forall (i,j) \in E, \\
& \quad h_i(x_i) = 0, \quad \forall i \in V, \\
& \quad g_i(x_i) \leq 0, \quad \forall i \in V, \\
& \quad \nu_{ij} \geq 0, \quad \forall (i,j) \in E,
\end{align*}
\]
\[ L_p(x, \theta) = \sum_{i=1}^{N} \left\{ f_i(x_i) + \sum_{j \in N_i} \left[ \nu_{ij}^T (x_i - x_j) + \frac{\rho_{ij}}{2} \| x_i - x_j \|^2 \right] + \lambda_i^T h_i(x_i) + \frac{\varrho_i}{2} \| h_i(x_i) \|^2 + 1^T \varphi_i(x_i) \right\} \]  

where the \( q \) operator is to be intended component-wise.

Notice that, more generally, one can associate a different penalty parameter to each component of the equality and inequality constraints of each node. This extension is omitted in order to streamline the presentation.

A powerful method for solving problem \( (2) \) is the well known Method of Multipliers, which consists of the following steps (see e.g. \[23, 19\]),

\[ x^{k+1} = \arg \min_x L_p^k(x, \theta^k) \]  

\[ \nu_{ij}^{k+1} = \nu_{ij}^k + \rho_{ij}^k (x_i^{k+1} - x_j^{k+1}) \], \quad \forall (i, i) \in \mathcal{E}, \quad \forall i \in \mathcal{V}, \quad \forall i \in \mathcal{V}, \]  

\[ \lambda_i^{k+1} = \lambda_i^k + g_i^k h_i(x_i^{k+1}) \], \quad \forall i \in \mathcal{V}, \quad \forall i \in \mathcal{V}, \]  

\[ \mu_i^{k+1} = \max(0, \mu_i^k + \zeta_i^k g_i(x_i^{k+1})) \], \quad \forall i \in \mathcal{V}, \quad \forall i \in \mathcal{V}, \]  

where the max operator is to be intended component-wise and \( p^{k+1} \geq p^k \geq \ldots \geq p^0 > 0 \). A typical update rule (see \[19\]) for a penalty parameter \( \rho \) associated to an equality constraint \( h(x) = 0 \) is

\[ \rho^{k+1} = \begin{cases} \beta \rho^k, & \text{if } \| h(x^{k+1}) \| > \gamma \| h(x^k) \|, \\ \rho^k, & \text{otherwise}, \end{cases} \]  

where \( \beta \) and \( \gamma \) are positive constants. Similarly, the update rule for a penalty parameter \( \zeta \) associated to an inequality constraint \( g(x) \leq 0 \) is

\[ \zeta^{k+1} = \begin{cases} \beta \zeta^k, & \text{if } \| g^+(x^{k+1}, \mu^{k+1}, \zeta^{k}) \| > \gamma \| g^+(x^k, \mu^k, \zeta^k) \|, \\ \zeta^k, & \text{otherwise}, \end{cases} \]  

where \( g^+(x, \mu, \zeta) = \max\{g(x), -\zeta\} \).

The minimization step \( (5) \) can be carried out approximately at each step \( k \), up to a certain precision \( \varepsilon^k \). If the sequence \( \{\varepsilon^k\} \to 0 \) as \( k \to \infty \), the minimization step is asymptotically exact (see \[19\] Section 2.5).

Sufficient conditions guaranteeing the convergence of method \( (5)-(8) \) to a local minimum of problem \( (2) \) have been given, e.g., in \[19\]. One of these conditions involves the regularity of the local minima of the optimization problem. In general, such a condition is not verified in problem \( (2) \) due to the constraints \( x_i = x_j \) for all \((i, j) \in \mathcal{E}\). In \[21, 22\], the results in \[19\] have been extended to deal with the non regularity of the local minima of problem \( (2) \).

It is worth stressing that the augmented Lagrangian defined in \( (4) \) is not separable in the local decision variables \( x_i \). Thus, the minimization step in \( (5) \) cannot be performed by independently minimizing with respect to each variable.
3 Asynchronous Method of Multipliers

In this section, the Asynchronous Method of Multipliers (ASYMM) for solving problem (2) in an asynchronous and distributed way is presented. Let us first present a distributed algorithm whose aim is to check whether all nodes in an asynchronous network have set a local flag to one. It can be seen as the asynchronous counterpart of the synchronous logic-AND algorithm presented in [24].

3.1 Asynchronous distributed logic-AND

Each node in the network is assigned a flag $C_i$ that is initially set to 0 and is then changed to 1 in finite time. The aim of the asynchronous distributed logic-AND algorithm is to check if all the nodes have $C_i = 1$.

Each node $i$ stores a matrix $S_i \in \{0, 1\}^{d_G \times d_i}$ which contains information about the status of the node itself and its neighbors. Let $S_i[l, j_i]$ denote the element in the $l$-th row and $j_i$-th column of $S_i$, where $j_i$ is the index associated to node $j$ by node $i$. The elements $S_i[1, j_i]$ for $j \in N_i$ represent the values of the flags of nodes $j \in N_i$ and $S_i[1, d_i]$ the one of node $i$ itself. This means that $S_i[1, d_i] = C_i$. Moreover, for $l = 2, \ldots, d_G$, the element $S_i[l, j_i]$, $j \in N_i$, contains the status of the $(l-1)$-th row of $S_j$, which is defined as the product of all its entries. Similarly, $S_i[l, d_i]$ contains the status of the $(l-1)$-th row of $S_i$ and it is computed as

$$S_i[l, d_i] = \prod_{b=1}^{d_i} S_i[l-1, b].$$

Hence, one has that $S_i[l, d_i] = 1$ if and only if $S_i[l-1, j_i] = 1$ for all $j \in N_i$ and $S_i[l-1, d_i] = 1$.

\[\text{Algorithm 1 Asynchronous distributed logic-AND}\]
\[\text{Initialization: } C_i \leftarrow 0, S_i \leftarrow 0_{d_G \times d_i}\]

\[\text{AWAKE}\]
\[\text{if } \prod_{b=1}^{d_i} S_i[d_G, b] \neq 1 \text{ then}\]
\[S_i[1, d_i] \leftarrow C_i\]
\[S_i[l, d_i] \leftarrow \prod_{b=1}^{d_i} S_i[l-1, b] \text{ for } l = 2, \ldots, d_G\]
\[\text{BROADCAST } S_i[\cdot, d_i] \text{ to all } j \in N_i\]

\[\text{if } \prod_{b=1}^{d_i} S_i[d_G, b] = 1 \text{ then}\]
\[\text{STOP and send STOP signal to all } j \in N_i\]

\[\text{IDLE}\]
\[\text{if } S_j[\cdot, d_j] \text{ received from } j \in N_i \text{ and not received a STOP signal then}\]
\[S_i[l, j_i] \leftarrow S_j[l, d_j] \text{ for } l = 1, \ldots, d_G\]

\[\text{if STOP received, set } S_i[d_G, \cdot] \leftarrow 1\]

A pseudo code of the distributed logic-AND algorithm is reported in Algorithm[1]. Notice, in particular, that node $i$ has to broadcast to all its neighbors only the last column of $S_i$, i.e. $S_i[l, d_i]$ for $l = 1, \ldots, d_G$. Moreover, it stores only the $d_j$-th column of the matrices $S_j$ of its neighbors $j \in N_i$, whenever it receives them.
It is apparent that a node will stop only when the last row of its matrix \( S_i \) is composed by all 1s, i.e. when
\[
\prod_{b=1}^{d_i} S_i[d_G, b] = 1. \tag{12}
\]

In the following result it is shown that (12) is satisfied at some node if and only if \( C_i = 1 \) for all \( i \).

**Proposition 1.** Let Assumption 3 hold. If there exists a time instant after which \( C_j=1 \) indefinitely for all \( j \in V \), then \( \prod_{b=1}^{d_i} S_i[d_G, b] = 1 \) in finite time for all nodes \( \ell \in V \). Conversely, if there exists a node \( \ell \) satisfying \( \prod_{b=1}^{d_i} S_i[d_G, b] = 1 \) at a certain time instant, then every node \( j \in V \) must have had \( C_j = 1 \) at some previous time instant.

**Proof.** In a time interval of \( \max_{i \in \{1, \ldots, N\}} T_i \), every node wakes up at least once. In the worst case, in which the distance between two generic nodes \( j \) and \( \ell \) is equal to the graph diameter \( (d_G) \), in a time \( d_G \max_{i \in \{1, \ldots, N\}} T_i \) there exists an ordered subsequence of awakenings following the path \( j \rightarrow \ell \). Hence, if \( C_j = 1 \) \( \forall \ell \), node \( h \) along this path will broadcast \( S_h[:,d_h] = 1 \) to its neighbors and \( S_{\ell}[d_{\ell}, :] \) will eventually contain only 1s, thus leading to \( \prod_{b=1}^{d_i} S_i[d_G, b] = 1 \). Now assume that \( \prod_{b=1}^{d_i} S_i[d_G, b] = 1 \) at some time instant and suppose, by contradiction, that there exists some node \( j \) for which \( C_j = 0 \) at all previous time instants. By assumption, all nodes \( i \in \mathcal{N}_j \) have \( S_i[1,j_i] = 0 \) (because columns sent by \( j \) always contained a zero in that position), which in turn, implies that \( S_i[2 : d_G, d_i] = 0 \) for all \( i \in \mathcal{N}_j \). Then, for every \( i \in \mathcal{N}_j \), every \( m \in \mathcal{N}_i \) have \( S_m[2, i_m] = 0 \) and hence \( S_m[3 : d_G, d_m] = 0 \). By induction, node \( \ell \) must have at least one element of its \( d_G \)-th row equal to 0, which contradicts the assumption and hence completes the proof. \( \square \)

### 3.2 Asynchronous distributed optimization algorithm

In order to present the algorithm for solving problem (2) it is useful to define a local augmented Lagrangian, whose minimization with respect to the decision variable \( x_i \) is equivalent to the minimization of the entire augmented Lagrangian (4). To this aim, let \( x_{\mathcal{N}_i} = [x_j]_{j \in \mathcal{N}_i \cup \{i\}}, \theta_{\mathcal{N}_i} = [\lambda_i, \mu_i, \nu_i, [\nu_j]_{j \in \mathcal{N}_i}], \) and \( p_{\mathcal{N}_i} = [\gamma_i, \xi_i, \rho_i, [\rho_{ji}]_{j \in \mathcal{N}_i}] \). Then, the \( i \)-th local augmented Lagrangian, which groups together all the terms of (4) depending on \( x_i \), is defined as
\[
\tilde{L}_{p_{\mathcal{N}_i}}(x_{\mathcal{N}_i}, \theta_{\mathcal{N}_i}) =
 = f_i(x_i) +
 + \sum_{j \in \mathcal{N}_i} \left[ x_i^\top (\nu_{ij} - \nu_{ji}) + \frac{\rho_{ij} + \rho_{ji}}{2} \|x_i - x_j\|^2 \right] +
 + \lambda_i^\top h_i(x_i) + \frac{\eta_i}{2} \|h_i(x_i)\|^2 +
 + 1^\top g_i, (\mu_i, g_i(x_i)). \tag{13}
\]

The following proposition holds.

**Proposition 2.** Let Assumption 7 hold. Then
\[
\nabla_{x_i} L_p(x, \theta) = \nabla_{x_i} \tilde{L}_{p_{\mathcal{N}_i}}(x_{\mathcal{N}_i}, \theta_{\mathcal{N}_i}). \tag{14}
\]
and for fixed values of \(x_j, j \neq i\)

\[
\arg\min_{x_i} \mathcal{L}_p(x, \theta) = \arg\min_{x_i} \tilde{\mathcal{L}}_{p_{N_i}}(x_{N_i}, \theta_{N_i}). \tag{15}
\]

Moreover, \(\tilde{\mathcal{L}}_{p_{N_i}}(x_{N_i}, \theta_{N_i})\) has Lipschitz continuous gradient for all \(i \in \mathcal{V}\).

**Proof.** From (4) and (13) it can be easily seen that

\[
\mathcal{L}_p(x, \theta) = \tilde{\mathcal{L}}_{p_{N_i}}(x_{N_i}, \theta_{N_i}) + \Psi(x_{-i}, \theta_{-i}) \tag{16}
\]

where \(\Psi(x_{-i}, \theta_{-i})\) is a function which does not depend on local variables of node \(i\). Hence (14) and (15) follow. By Assumption 1 it holds that \(\nabla_x \mathcal{L}_p(x, \theta)\) is almost everywhere differentiable for all \(\theta\) and \(p > 0\) (see e.g. [19], Proposition 3.1). Hence, one has that \(\nabla_x \mathcal{L}_p(x, \theta)\) has Lipschitz continuous gradient for all \(\theta\) and \(p > 0\). Moreover, from Assumption 1 it holds that \(\nabla_x \mathcal{L}_p(x, \theta)\) is bounded, and \(\mathcal{L}_p(x, \theta)\) has Lipschitz continuous gradient for all \(\theta\) and \(p > 0\). Hence \(\mathcal{L}_p(x, \theta)\) has block component-wise Lipschitz continuous gradients and, from (14) \(\tilde{\mathcal{L}}_{p_{N_i}}(x_{N_i}, \theta_{N_i})\) has Lipschitz continuous gradient.

The ASYMM algorithm for solving problem (2) in an asynchronous and distributed way is now introduced. The rationale of the algorithm is the following. When a node wakes up, it performs a gradient descent step on its local augmented Lagrangian until every node has reached a suitable accuracy. This check is performed by nodes themselves in a distributed way, using the logic-AND algorithm presented in Section 3.1. When a node gets aware of this condition, it performs one ascent step on its local multiplier vector. After it has received the updated multipliers from all its neighbors, it gets back to the primal update.

More formally, when node \(i\) wakes up, it checks through a flag, called \(M_{\text{done}}\), if its multiplier vector and the neighboring ones are up to date. If this is the case (which corresponds to \(M_{\text{done}} = 0\)), it performs one of the following two tasks:

**T1.** If \(\prod_{i=1}^d S_i[d_G, l] \neq 1\), node \(i\) performs a gradient descent step on its local augmented Lagrangian (using \(1/L_i\) as stepsize, where \(L_i\) is the Lipschitz constant of \(\tilde{\mathcal{L}}_{p_{N_i}}(x_{N_i}, \theta_{N_i})\)) and checks if the local tolerance \(\epsilon_i > 0\) on the gradient has been reached. If the latter is true, it corresponds to setting \(C_i \leftarrow 1\) in the distributed logic-AND Algorithm 1. Then, it updates matrix \(S_i\), and broadcasts the updated \(x_i\) and the column \(S_i[:, d_i]\) to its neighbors.

**T2.** If \(\prod_{i=1}^d S_i[d_G, l] = 1\), node \(i\) performs an ascent step on the local multiplier vector and updates the local penalty parameters according to equations (6-8) and (9-10), respectively. Then, it sets \(M_{\text{done}} = 1\) and broadcasts the updated multipliers and penalty parameters \(\nu_{ij}\) and \(\rho_{ij}\) (associated to constraints \(x_i = x_j\)) to its neighbors.

![Figure 1: An example of the execution of ASYMM for a network with three nodes.](image-url)
Algorithm 2 ASYMM

Initialization: Initialize $x_i, \theta_i, N_i, p_i, S_i = 0_{d_G \times d_i}, M_{done} = 0$.

**AWAKE**
if $\prod_{b=1}^{d_i} S_i[d_G, b] \neq 1$ and not $M_{done}$ then

$$x_i \leftarrow x_i - \frac{1}{L_i} \nabla x_i \tilde{L}_{p_{N_i}}(x_{N_i}, \theta_{N_i})$$
if $\|\nabla x_i \tilde{L}_{p_{N_i}}(x_{N_i}, \theta_{N_i})\| \leq \epsilon_i$ then $S_i[1, d_i] \leftarrow 1$
$S_i[l, d_i] \leftarrow \prod_{b=1}^{d_i} S_i[l - 1, b]$ for $l = 2, ..., d_G$

**BROADCAST** $x_i, S_i[; : d_i]$ to all $j \in N_i$

if $\prod_{b=1}^{d_i} S_i[d_G, b] = 1$ and not $M_{done}$ then

$$\nu_{ij} \leftarrow \nu_{ij} + \rho_{ij}(x_i - x_j) \text{ for } j \in N_i$$
$\lambda_i \leftarrow \lambda_i + g_i h_i(x_i)$
$\mu_i \leftarrow \max\{0, \mu_i + \zeta_i g_i(x_i)\}$

update $g_i, \zeta_i$ and $\rho_i$

$M_{done} \leftarrow 1$

**BROADCAST** $\nu_{ij}, \rho_{ij}$ to $j \in N_i$

**IDLE**
if $S_j[; : d_j]$ received from $j \in N_i$ and not already received some new $\nu_{ji}$ then

$S_j[l, j] \leftarrow S_j[l, d_j]$ for $l = 1, ..., d_G$
if $\nu_{ji}$ and $\rho_{ji}$ received from $j \in N_i$ set $S_i[d_G, :] \leftarrow 1$
if $x_{new}$ received from $j \in N_i$, update $x_j \leftarrow x_{new}$
if $M_{done}$ and $\nu_{ji}$ received from all $j \in N_i$ then

$M_{done} \leftarrow 0, S_i \leftarrow 0_{d_G \times d_i}, \text{update } \epsilon_i$
When in \textit{IDLE}, node \(i\) continuously listens for messages from its neighbors, but does not broadcast any information. Received messages may contain either local optimization and logic-AND variables, or multiplier vectors and penalty parameters. If necessary, node \(i\) suitably updates local logic-AND variables or the \(M_{\text{done}}\) flag. Notice that, for node \(i\), sending a new multiplier \(\nu_{ij}\) or receiving a new \(\nu_{ji}\) corresponds to sending or receiving a \textit{STOP} signal in the asynchronous logic-AND algorithm.

The ASYMM pseudocode is reported in Algorithm 2 and an example of its execution is shown in Figure 1, where tasks \(T_1\) and \(T_2\) are denoted by white and black blocks, respectively.

\section{ASYMM Analysis}

In order to analyze the ASYMM algorithm, we start by noting that under Assumptions 3 and 4, from a global perspective, the local asynchronous updates can be treated as an algorithmic evolution in which, at each iteration, only one node wakes up in an essentially cyclic fashion.

Given the above, it is possible to associate an iteration of the distributed algorithm to each triggering. Denote by \(t \in \mathbb{N}\) a discrete, universal time indicating the \(t\)-th iteration of the algorithm and define as \(i_t \in \mathcal{V}\) the index of the node triggered at iteration \(t\).

In the following, it will be shown that: (i) there is an \textit{equivalence} relationship between ASYMM and an inexact Method of Multipliers and (ii) \textit{convergence} of ASYMM to a local minimum of problem (1) is guaranteed under suitable conditions inherited from the centralized optimization literature.

\subsection{Equivalence with an inexact Method of Multipliers}

Consider an inexact Method of Multipliers which consists of solving the \(k\)-th instance of the augmented Lagrangian minimization by means of a block-coordinate gradient descent algorithm (see, e.g., [25] for a survey), which runs for a certain number of iterations \(h^k\). A pseudo code of this inexact Method of Multipliers is given in Algorithm 3, where \(i_h\) is the index of the block chosen at iteration \(h\) and the penalty parameters are updated as in (9)-(10).

\begin{algorithm}
\caption{Inexact MM}
\begin{algorithmic}
\FOR {\(k = 0, 1, \ldots\)}
\STATE \(\hat{x}^0 = x^k\)
\FOR {\(h = 1, \ldots, h^k\)}
\STATE \(\hat{x}^{h+1} = \hat{x}^h - \frac{1}{L_{ih}} \nabla_x \mathcal{L}_p(\hat{x}^h, \theta^k)\)
\STATE \(x^{k+1} = \hat{x}^{h+1}\)
\STATE \(\nu_{ij}^{k+1} = \nu_{ij}^k + \rho_{ij}^k (x_i^{k+1} - x_j^{k+1}), \quad \forall (i,j) \in \mathcal{E}\)
\STATE \(\lambda_{i}^{k+1} = \lambda_{i}^k + \rho_{i}^k h_i(x_i^{k+1}), \quad \forall i \in \mathcal{V}\)
\STATE \(\mu_i^{k+1} = \max\{0, \mu_i^k + \zeta_i^k g_i(x_i^{k+1})\}, \quad \forall i \in \mathcal{V}\)
\ENDFOR
\ENDFOR
\end{algorithmic}
\end{algorithm}

It is worth remarking that the ordered sequence of indexes \(h\) and \(k\) used in Algorithm 3 does not coincide with the sequence of universal times \(t\) of ASYMM. It will be rather shown that a (possibly reordered) subsequence of iterations in the universal time \(t\) of ASYMM gives rise to suitable \(h\) and \(k\) sequences in Algorithm 3.
Let \( t_1, t_2, \ldots \) be a subsequence of \( \{t\} \) such that at each \( t_k \) a multiplier update (task T2) has been performed by node \( i_k \) and let \( t_1 \) be the time instant of the first multiplier update. Then, the following result holds.

**Lemma 3.** Each sequence \( (i_{tkN+1}, \ldots, i_{tkN+k}) \), for \( k = 0, 1, \ldots \), is a permutation of \( \{1, \ldots, N\} \). Moreover, if \( \epsilon_i > 0 \) for all \( i \in \mathcal{V} \), multiplier updates occur infinitely many times.

**Proof.** Consider the first multiplier update, performed by node \( i_1 \) at \( t_1 \). Then, until all other nodes have performed their first multiplier update, there will be some node \( j \) which has not received back all the new multipliers from its neighbors and has \( M_{\text{done}} = 1 \). Hence, it cannot run task T1, and consequently it cannot set and broadcast \( S_j[1, d_j] = 1 \). So, node \( i_1 \) has at least one element of the last row of \( S_i \) at 0, hence it cannot perform another multiplier update (although it could have started over performing task T1). The first part of the proof is completed by induction. In order to prove the second part of the lemma, assume, by contradiction, that the number of multiplier updates is finite and denote by \( t_M \) the time instant of the last multiplier update. If \( \text{mod}(M, N) \neq 0 \), from the connectedness of \( \mathcal{G} \) at \( t_M + 1 \) there exists at least one node \( j \) that: i) has not updated its multipliers yet; ii) has a neighbor who has already performed a multiplier update. Hence, node \( j \) will perform a multiplier update next time it wakes up (which occurs in finite time by Assumption 3), thus contradicting the assumption that \( t_M \) was the time instant of the last multiplier update. If \( \text{mod}(M, N) = 0 \) (i.e., \( (i_{tkN}, \ldots, i_{tkN+k}) \) is a permutation of \( \{1, \ldots, N\} \)), all the nodes that wake up after \( t_M \) will run task T1. From a global perspective, this can be seen as a block coordinate descent algorithm on the augmented Lagrangian with a given multiplier vector. Since this algorithm converges to a stationary point [20], every node \( i \in \mathcal{V} \) will reach its local tolerance \( \epsilon_i > 0 \) in finite time and then set \( S_i[1, d_i] = 1 \). From Proposition 1, after a finite number of iterations some node \( j \) will satisfy \( \prod_{b=1}^{G} S_j[\ell, b] = 1 \) and hence it will run a new multiplier update, which contradicts the assumption that \( t_M \) was the time instant of the last multiplier update.

In the sequel the subset of universal times \( t, \{t_{kN+1}, \ldots, t_{(k+1)N}\} \), during which \( N \) tasks T2 are performed, will be referred to as the \( k \)-th cycle of ASYMM.

The example in Figure 1 shows the \( k \)-th and \( (k+1) \)-th cycles of an ASYMM run. According to Lemma 3 during a single cycle each node performs task T2 once. It is worth remarking that in the \( k \)-th cycle node 1 starts over the \( (k+1) \)-th primal minimization before node 3 has completed its \( k \)-th multiplier update. This happens because node 1 has already received the updated multipliers and penalty parameters from node 2, which is the only neighbor of node 1. The same thing happens to node 3 in the \( (k+1) \)-th cycle. This is a key feature of the asynchronous distributed scheme underlying ASYMM. It can also be observed that when node 3 wakes up for the first time, after node 1 has done the \( k \)-th dual update, it performs again task T1. In fact, node 3 has not received a STOP signal yet, because it is not connected directly to node 1.

Define \( \theta_i = [\lambda_i, \mu_i, \nu_i] \) and let \( \hat{x}_i \) and \( \hat{A}_i \) be the value of the state vector and of the multiplier vector of node \( i \), at iteration \( t \), computed according to ASYMM. Then, the following Corollary holds, whose proof follows immediately from Lemma 3.

**Corollary 4.** Let \( \tau \in \{t_{kN+1}, \ldots, t_{(k+1)N}\} \), for some \( k = 0, 1, \ldots \). Then one has \( \hat{\theta}_{\tau_{kN+t}} = \hat{\theta}_{\tau_{kN+t}} \) for all \( t = \tau, \tau+1, \ldots, t_{(k+1)N} \).

Corollary 4 states that once a multiplier vector is updated in a cycle, then, its value remains unchanged for the whole cycle.
Let $\tau_i^k$ be the time instant in which node $i$ performs task $T2$ in the $k$-th cycle, i.e., $\tau_i^k \in \{t_{kN+1}, \ldots, t_{(k+1)N}\}$ such that node $i$ is awake at time $\tau_i^k$. By using this Corollary 4 one can define

$$x_i^{k+1} = x_i^k, \quad \theta_i^{k+1} = \theta_i^k,$$

$k = 0, 1, \ldots$ and by using Lemma 3 and reordering the indexes \(i\),

$$x^{k+1} = [(x_1^{k+1})^T, \ldots, (x_N^{k+1})^T]^T,$$

$$\theta^{k+1} = [(\theta_1^{k+1})^T, \ldots, (\theta_N^{k+1})^T]^T.$$

The next two lemmas show that a local primal (resp. multiplier) update is performed according to a common multiplier (resp. primal) variable.

**Lemma 5.** For all $\tau \in \{t_{kN+1}, \ldots, t_{(k+1)N}\}$, $k = 0, 1, \ldots$, every multiplier $\theta^{k+1}_v$ is computed using the state vector $x^{k+1}$.

**Proof.** Consider $\tau \in \{t_{kN+1}, \ldots, t_{(k+1)N}\}$ for a given $k$. Node $i$, computes

$$x_i^{k+1} = x_i^\tau, \quad \nu_i^{k+1} = \nu_i^\tau + \rho_i x_i^{k+1} - x_i^\tau, \quad \forall j \in N_i,$$

$$\lambda_i^{k+1} = \lambda_i^\tau + \theta_i h_i(x_i^{k+1}),$$

$$\mu_i^{k+1} = \max\{0, \mu_i^\tau + \gamma_i t_i(x_i^{k+1})\}.$$

First, notice that the update of node $i$ depends only on $x_j$ or $\theta_j$, with $j \in N_i$. Then, let us show that $\tilde{x}_j^\tau = x_j^{k+1}$ for all $j \in N_i$. If a node $j \in N_i$ has already performed its multiplier update in the $k$-th cycle, then, even if it woke up again before time $\tau$, it did not update $x_j$ (did not start a new primal update) because it has not received all the updated multipliers from its neighbors (node $i$, has not performed the multiplier update yet and thus has not sent $\nu_i^{k+1}$ to node $j$). Therefore, $\tilde{x}_j^\tau = x_j^{k+1}$. If, vice-versa, node $j \in N_i$, has not performed its multiplier update in the $k$-th cycle, then $\tilde{x}_j^\tau$ will become $x_j^{k+1}$ next time node $j$ wakes up (because node $i$, has sent it the updated multiplier while it was in idle, so that $j$ has set $S_j[dG, \cdot] = 1$).

**Lemma 6.** For all $t = \tau_i^k, \tau_i^{k+1}, \ldots, \tau_i^{k+1}$, every descent step on the augmented Lagrangian with respect to the block-coordinate $x_i$ is performed using the multiplier vector $\theta^{k+1}_i$.

**Proof.** Node $i$ can start over a block coordinate descent iteration after $\tau_i^k$ only when it has received all the new multipliers $\nu_i^{k+1}$ (and the corresponding penalty parameters) from its neighbors. Thanks to (14), it is sufficient to show that each local descent on the $i$-th local augmented Lagrangian is performed using the multiplier vector $\theta^{k+1}_{x_i}$. This follows from Lemma 3 and Corollary 1 by using arguments similar to those in the proof of Lemma 5.

Next lemma states that every node performs at least one primal update between the beginning of two consecutive cycles.

**Lemma 7.** Between $t_{kN+1}$ and $t_{(k+1)N+1}$ every node performs task $T1$ at least once.

**Proof.** Since at iteration $t_{(k+1)N}$ all the nodes have performed the $k$-th multiplier update, each of them has set $S_i[1, d_i] = 0$ at some time between $t_{kN+1}$ and $t_{(k+1)N}$. At time $t_{(k+1)N+1}$ the first node performs the $(k+1)$-th multiplier update. This can occur only if each node $i$ has set $S_i[1, d_i] = 1$ at some time between $t_{kN+1}$ and $t_{(k+1)N+1}$, which implies that each node performs task $T1$ at least once over the same time interval.
The equivalence of ASYMM and Algorithm 3 is stated by the next theorem.

**Theorem 8.** Let Assumptions 3 and 4 hold. Then, ASYMM is equivalent to an instance of Algorithm 3 in which the selection of nodes $i_h$ satisfies an essentially cyclic rule. Moreover, if in Algorithm 2, $\epsilon_i > 0$, $\forall i \in V$, the total number of primal descent steps $h^k$ is finite.

**Proof.** Define $H^k = \{ t : \tau_{i-1}^k < t < \tau_i^k, i \text{ runs task T1 at } t \}$ for $k = 0, 1, ..., \infty$, where $\tau_i^k$ is the first time instant in which node $i$ is awake (doing task T1). Then, define $H^k = \bigcup_{i \in V} H_i^k$, $h^k = |H^k|$ and let $v_1 < v_2 < ... < v_h^k$

be the ordered sequence of elements (time instants) in $H^k$.

By setting $i_h = i_{v_h}$, for $h = 1, \ldots, h^k$, in Algorithm 3 and using Lemma 5 and Lemma 6, one has that ASYMM turns out to be equivalent to an instance of Algorithm 3. Moreover, from Lemma 7 every node runs task T1 at least once in $\{v_1, \ldots, v_{h^k}\}$. Hence, Algorithm 3 is run with an essentially cyclic update rule over a time window of length $h^k$, which is finite due to Lemma 3.

**Remark 1.** The duration of each ASYMM cycle is bounded. From the definition of $h^k$ it can be easily verified that $h^k \geq N$. Moreover, from Lemma 7 one has that $h^k \leq t_{k+1}N + 1 - t_{kN} - N$. Hence, $t_{k+1}N + 1 - t_{kN} \geq 2N$. On the other side, from Proposition 4 it follows that $t_{k+1}N - t_{kN} \leq \max_i T_i$ for all $k$.

**Remark 2.** Assumption 4 can be relaxed, allowing for non neighboring nodes to be awake in the same time instant, without affecting the convergence equivalence of ASYMM.

### 4.2 Local Convergence

Let us first introduce a result that allows us to bound the norm of the gradient of a strongly convex function with block component-wise Lipschitz continuous gradient, during the evolution of a block coordinate descent algorithm. Specifically, we relate this bound to given tolerances $\epsilon_i$ on the norm of the gradient with respect to block $i$.

**Lemma 9.** Let $\Phi(y_1, \ldots, y_N)$ be a $\sigma$-strongly convex function with block component-wise Lipschitz continuous gradients (with $L_i$ being the Lipschitz constant with respect to block $y_i$) in a subset $Y \subseteq \mathbb{R}^n$. Let $\{y^h\}$ be a sequence generated according to $y^{h+1} = y^h - \frac{1}{L_i} U_{i_h} \nabla y_{i_h} \Phi(y^h)$, where $y^0 \in Y$ and indexes $i_h \in \{1, \ldots, N\}$ are drawn in an essentially cyclic way. If, for some $h > 0$, $\|\nabla y_i \Phi(y^h)\| \leq \epsilon_i$, $\forall i \in \{1, \ldots, N\}$, then

$$\|\nabla y_i \Phi(y^h)\| \leq \sqrt{\frac{\sum_{i=1}^N \left( \frac{L_i \epsilon_i}{\sigma} \right)^2}{N}}$$

for all $h \geq \tilde{h}$.

**Proof.** The proof is given in the Appendix.

In order to show the local convergence, an additional assumption is needed.

**Assumption 5.** There exists some $\tilde{k} > 0$, such that for all $k \geq \tilde{k}$, the sequence $x^k$ generated by ASYMM belongs to a neighborhood of a local minimum of $L_{p_k}(x, \theta^k)$ where $L_{p_k}(x, \theta^k)$ is $\sigma^k$-strongly convex.

13
Assumption 5 is indeed strong, but it is somehow standard in the optimization literature. As pointed out, e.g., in [19], while it is not possible to guarantee that such an assumption holds a priori, in practice it turns out to be typically satisfied after a sufficient number of iterations of the primal minimization and multiplier/penalty parameter update. The next theorem shows that ASYMM guarantees $\|\nabla_x L_p^k(x^{k+1}, \theta^k)\| \leq \varepsilon^k$, with $\varepsilon^k$ depending on the local thresholds $\epsilon_i$.

**Theorem 10.** Let Assumptions 1, 2 and 5 hold. Denote by $\epsilon^k_i$ the local tolerance set by node $i$ for the primal descent during cycle $k$ and by $L^k_i$ the Lipschitz constant of $\nabla_x L_p^k(x, \theta^k)$ with respect to $x_i$. Then, there exists $\bar{k} > 0$, such that for all $k \geq \bar{k}$, it holds $\|\nabla_x L_p^k(x^{k+1}, \theta^k)\| \leq \varepsilon^k_i$ with

$$\varepsilon^k = \sqrt{\sum_{i=1}^{N} \left( \frac{L^k_i \epsilon^k_i}{\sigma^k} \right)^2}.$$  

Proof. From Proposition 2, $L_p^k(x, \theta^k)$ has block component-wise Lipschitz continuous gradient with constants $L^k_i$.

Moreover, at $v_h$, from $\|\nabla_x \tilde{L}_{p,v_i}(x_{N_i}, \theta_{N_i})\| \leq \epsilon^k_i$ and (14), one has that under Assumption 5 for $k \geq \bar{k}$

$$\|\nabla_x L_p^k(x^{k+1}, \theta^k)\| \leq \epsilon^k_i, \forall i \in V$$

and hence, from Lemma 9,

$$\|\nabla_x L_p^k(x^{k+1}, \theta^k)\| \leq \varepsilon^k = \sqrt{\sum_{i=1}^{N} \left( \frac{L^k_i \epsilon^k_i}{\sigma^k} \right)^2},$$

thus concluding the proof.

We want to remark that the only global parameter $\sigma^k$ appearing in our analysis does not need to be known by the nodes, because it is not required for the execution of ASYMM. In fact, once (17) is satisfied, (18) also automatically holds.

We stress that ASYMM is equivalent to Algorithm 3 even without Assumption 5, which is needed only to guarantee the local convergence to a (strict) local minimum of problem (1). In fact, for $k < \bar{k}$ in Assumption 5 the augmented Lagrangian can be nonconvex, thus Lemma 9 cannot be invoked. Nevertheless, the block coordinate descent algorithm is still guaranteed to converge (at least) to a stationary point as shown in [20]. This means that multiplier updates in ASYMM will surely occur after a finite number of primal steps. Moreover, as $p^k$ grows, the augmented Lagrangian typically becomes locally strongly convex, see, e.g., [19]. If this happens, the block coordinate descent algorithm approaches the corresponding minimum of the augmented Lagrangian and, provided that for $k \geq \bar{k}$ the local tolerances $\epsilon^k_i$ vanish as $k \to \infty$, the minimization of the augmented Lagrangian is asymptotically exact. This guarantees a convergence result for ASYMM in the same sense as the one in the centralized case: the convergence of the algorithm is contingent upon the generation of (possibly local) minima of the augmented Lagrangian that, after some index $\bar{k}$, stay in the neighborhood of the same local minimum $x^*$ of problem (1). As reported in [19], extensive numerical experience has shown that, from a practical point of view, choosing the obtained $x^k$ as the initial condition for the $(k+1)$-th minimization usually generates sequences $\{x^k\}$ within a neighborhood of the same local minimum $x^*$. 

14
5 Dealing with big-data optimization

The ASYMM algorithm can be easily amended to deal with so called distributed big-data optimization, i.e., the distributed solution of problems in which \( x \in \mathbb{R}^n \) with \( n \) very large. In this set up arising, e.g., in estimation and learning problems, two main problems may arise. On one hand, the primal and multiplier update steps may not be executable in a single step by some node because the computation of the whole gradient in the primal step may be cumbersome. Moreover, communication bottlenecks may arise, in fact it may happen for some node \( i \), that the local optimization variable \( x_i \) does not fit the communication channels between node \( i \) and its neighbors.

Assume each agent \( i \) to partition its local decision variable \( x_i \) in \( N_i \) blocks, i.e., \( x_i = [x_{i,1}, \ldots, x_{i,N_i}]^T \), where \( x_{i,m} = U_{i,m}^\top x_i \) and \( x_i = \sum_{m=1}^{N_i} U_{i,m} x_{i,m} \).

Whenever node \( i \) wakes up to perform task T1, it computes the primal descent step on one of the \( N_i \) blocks (say \( m \)) instead of performing it on the whole \( x_i \), i.e. it computes

\[
 x_{i,m} = x_{i,m} - \frac{1}{L_{i,m}} \nabla x_{i,m} \tilde{L}_p(x_{N_i}, \theta_{N_i})
\]

where \( m \) is picked in an essentially cyclic way. Similarly the update of the local multiplier vectors can be carried out on one block at a time, e.g.,

\[
 \nu_{ij,m} \leftarrow \nu_{ij,m} + \rho_{ij}(x_{i,m} - x_{j,m}).
\]

By following the same reasoning adopted in Sections 3 and 4, it can be shown that the primal descent steps are equivalent to a block coordinate descent algorithm on the augmented Lagrangian and converge to a stationary point.

The only additional assumption needed for the convergence result is that functions \( f_i \), \( h_i \), \( g_i \) have block component-wise Lipschitz continuous gradients. If for some node \( i \), \( x_i \) does not fit the communication channels (the same holds true for \( \nu_{ij} \)), ASYMM is easily extended by allowing node \( i \) to transmit \( x_{i,m} \) and \( U_{i,m} \) at the end of each task T1 and to split the transmission on \( \nu_{ij} \) in multiple steps.

6 Numerical Results

Two examples are presented to assess the performance of ASYMM. The first one involves nonconvex local constraints, while the second requires the minimization of a nonconvex objective function.

6.1 Distributed source localization

Consider a network of \( N \) sensors, deployed over a certain region, communicating according to a connected graph \( G = (\mathcal{V}, \mathcal{E}) \), which have to solve the optimization problem

\[
 \begin{align*}
 \text{minimize} & \quad \sum_{i=1}^{N} f_i(x) \\
 \text{subject to} & \quad \|x - c_i\| - R_i \leq 0, \quad i = 1, \ldots, N \\
 & \quad r_i - \|x - c_i\| \leq 0, \quad i = 1, \ldots, N,
\end{align*}
\]

which can be rewritten in the form of problem (2).
Such a problem naturally arises, for example, in the context of source localization, in which each agent knows its own absolute location $c_i$ and takes a noisy measurement $y_i$ of its own distance from an emitting source located at an unknown location $x^*$ (for example through a laser) as $y_i = \|x^* - c_i\| + w_i$. If we make the assumption of unknown but bounded (UBB) noise, i.e. for all $i = 1, \ldots, N$, the noise signal $w_i$ satisfies $|w_i| \leq \kappa_i$ for some $\kappa_i \geq 0$, then each node is able to define its own feasible set $X_i$ in which the unknown source location must lie as $X_i = \{x \mid r_i \leq ||x - c_i|| \leq R_i\}$, where $r_i = y_i - \kappa_i$ and $R_i = y_i + \kappa_i$. Notice that each set $X_i$ is a circular crown and hence it is a non convex set.

Suppose $f_i(x_i) = x_i^T x_i$ for all $i \in V$. We report a simulation with $N = 10$ nodes and $n = 2$, in which $x^* \in U[-2.5, 2.5]^n$, $c_i \in U[-2.5, 2.5]^n$ and $\kappa_i = U[0, 0.3]$ for all $i \in V$, where $U[a, b]$ denotes the uniform distribution between $a$ and $b$. The graph is modeled through a connected Watts-Strogatz model in which nodes has mean degree $K = 2$. Let us define the measure of infeasibility at iteration $k$ as

$$\xi^k = \sum_{i=1}^{N} \left( \max(0, ||x_i^k - c_i|| - R_i) + \max(0, r_i - ||x_i^k - c_i||) + \sum_{j \in N_i} ||x_i^k - x_j^k|| \right)$$

We run ASYMM for 25000 iterations with $\beta = 4$ and $\gamma = 0.25$. Figure 2a shows the evolution of $x^k_i$ for each $i \in V$. Finally, in Figure 2b the values of $\xi^k$ are reported. As it can be seen, the nodes performed 50 multiplier updates each, along the 25000 iterations (corresponding to 2500 awakenings per node on average).

### 6.2 Distributed nonlinear classification

In this example we consider a nonlinear classification problem in which the data to be classified are represented as points $z \in \mathbb{R}^2$ which belong to two different classes. So, each point is associated a label $y \in \{-1, 1\}$, which represents the class the point belongs to.

The considered classifier can be represented as a Neural Network (NN) consisting of one input layer with two units, two hidden layers with four and two units respectively, and an output layer with one unit (respectively green, blue and red in Figure 3). Moreover, a bias unit is present in both the input and the hidden layers (in yellow in Figure 3). Define
Define the output of the first hidden layer as
\[
l_1(z, w_1, b_1) = \tanh(w_1^T z + b_1),
\]
where the operator \( \tanh \) is to be intended component-wise. Similarly, the output of the second hidden layer is
\[
l_2(z, w_1, b_1, w_2, b_2) = \tanh(w_2^T l_1(z, w_1, b_1) + b_2)
\]
and the output of the whole NN is
\[
f(z, x) = \tanh(w_3^T l_2(z, w_1, b_1, w_2, b_2) + b_3)
\]
Given a set of labeled data points \((Z, Y)\), the classification problem can be written as
\[
\minimize_x \sum_{z, y \in (Z, Y)} (f(z, x) - y)^2.
\]
Suppose now that the dataset is distributed among \(N\) nodes, which communicate according to a connected graph \(G = (V, E)\). Each node \(i\) owns a portion of the dataset \((Z_i, Y_i)\), which must remain private and cannot be shared with the other nodes. In this framework, problem \(22\) can be rewritten in the equivalent form
\[
\minimize_{x_1, \ldots, x_N} \sum_{i=1}^{N} \sum_{z, y \in (Z_i, Y_i)} (f(z, x_i) - y)^2
\]
subject to
\[
x_i = x_j, \quad \forall (i, j) \in E
\]
In our simulations two datasets are considered, which are benchmarks used in the context of machine learning. The first one consists of points belonging to two moon-shaped subsets.
Figure 4: Red dots represent points with label $-1$ and blue dots points with label $1$. Colored regions represent the output of the classifier (21) resulting from the solution of (23) provided by ASYMM. The color of the regions is associated to a number as in the color bar.

(see Figure 4a), in which points of one subset have label $y = 1$, points of the other have label $y = -1$. In the second one, data points are distributed along two nested circles. Points on the inner circle have label $y = 1$, while the others have label $y = -1$ (see Figure 4b).

The ASYMM algorithm has been run on $N = 10$ nodes, each one processing a local dataset $(Z_i, Y_i)$ consisting of 100 points. The obtained classifiers are represented in Figures 4a and 4b. The colored regions represent the value of (21) computed in those points at the (local) minimum $x^*$ obtained by ASYMM.

It is worth stressing that the example presents a low-size classification problem, with the purpose of illustrating the proposed technique. When massive data in higher dimensional spaces are available, it is necessary to consider a more complex neural network (i.e., with a much higher number of neurons) so that the dimension of the decision variable can be fairly high. In such a case, the big-data approach proposed in Section 5 can be adopted.

7 Conclusions

In this paper, an asynchronous distributed algorithm for nonconvex optimization problems over networks has been proposed. By suitably defining local augmented Lagrangian functions, the optimization process can be distributed among the agents of the network. A fully asynchronous implementation has been devised, that takes advantage of a distributed logic-AND algorithm allowing agents to regulate the sequence of primal and dual update steps. The proposed ASYMM algorithm is shown to be equivalent to an inexact version of the centralized method of multipliers, thus inheriting its main properties. Under the same technical assumptions typically made in the centralized setting, the proposed algorithm is proven to converge to a stationary point of the augmented Lagrangian. Finally, an extension to big-data problems, featuring high-dimensional decision variables, has been presented.

Ongoing research concerns the specialization of the proposed method to different application domains, including distributed set membership estimation and machine learning with constraints.
References

[1] P. Bianchi and J. Jakubowicz, “Convergence of a multi-agent projected stochastic gradient algorithm for non-convex optimization,” IEEE Transactions on Automatic Control, vol. 58, no. 2, pp. 391–405, 2013.

[2] H.-T. Wai, A. Scaglione, J. Lafond, and E. Moulines, “A projection-free decentralized algorithm for non-convex optimization,” in Signal and Information Processing (GlobalSIP), 2016 IEEE Global Conference on. IEEE, 2016, pp. 475–479.

[3] P. Di Lorenzo and G. Scutari, “Next: In-network nonconvex optimization,” IEEE Transactions on Signal and Information Processing over Networks, vol. 2, no. 2, pp. 120–136, 2016.

[4] Y. Sun, G. Scutari, and D. Palomar, “Distributed nonconvex multiagent optimization over time-varying networks,” in Signals, Systems and Computers, 2016 50th Asilomar Conference on. IEEE, 2016, pp. 788–794.

[5] T. Tatarenko and B. Touri, “Non-convex distributed optimization,” IEEE Transactions on Automatic Control, vol. 62, no. 8, pp. 3744–3757, 2017.

[6] S. Lee and A. Nedic, “Distributed random projection algorithm for convex optimization,” IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 2, pp. 221–229, 2013.

[7] K. Margellos, A. Falsone, S. Garatti, and M. Prandini, “Distributed constrained optimization and consensus in uncertain networks via proximal minimization,” IEEE Transactions on Automatic Control, vol. PP, no. 99, pp. 1–1, 2017.

[8] P. Lin, W. Ren, and Y. Song, “Distributed multi-agent optimization subject to non-identical constraints and communication delays,” Automatica, vol. 65, pp. 120 – 131, 2016.

[9] I. Necoara, “Random coordinate descent algorithms for multi-agent convex optimization over networks,” IEEE Transactions on Automatic Control, vol. 58, no. 8, pp. 2001–2012, 2013.

[10] C.-X. Shi and G.-H. Yang, “Augmented lagrange algorithms for distributed optimization over multi-agent networks via edge-based method,” Automatica, vol. 94, pp. 55 – 62, 2018.

[11] A. Falsone, K. Margellos, S. Garatti, and M. Prandini, “Dual decomposition for multi-agent distributed optimization with coupling constraints,” Automatica, vol. 84, pp. 149 – 158, 2017.

[12] E. Wei and A. Ozdaglar, “On the O(1=k) convergence of asynchronous distributed alternating direction method of multipliers,” in Global conference on signal and information processing (GlobalSIP), 2013 IEEE. IEEE, 2013, pp. 551–554.

[13] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, “Explicit convergence rate of a distributed alternating direction method of multipliers,” IEEE Transactions on Automatic Control, vol. 61, no. 4, pp. 892–904, 2016.
Appendix

Proof of Lemma 9

Consider a function $\Phi(y)$, $\sigma$-strongly convex in a subset $Y \subseteq \mathbb{R}^n$, with $L$-Lipschitz continuous gradient. Define $y^* = \arg\min_{y \in Y} \Phi(y)$. From the definition, if $\Phi(y)$ is $\sigma$-strongly convex,
then, using the Cauchy Schwartz inequality one obtains
\[ \|\nabla \Phi(y) - \nabla \Phi(z)\| \geq \sigma \|y - z\|. \]

Then, it can be easily proved that
\[ \sigma \|y - y^\star\| \leq \|\nabla \Phi(y)\| \leq L\|y - y^\star\|, \ \forall y \in Y \]

In order to prove the following technical results (whose proof is reported for completeness) are needed.

**Lemma 11.** Performing a gradient descent algorithm on \( \Phi(y) \), starting from \( y^0 \) and using a step-size equal to \( \frac{1}{L} \), i.e.
\[ y^{h+1} = y^h - \frac{1}{L} \nabla_y \Phi(y^h) \]
produces a sequence \( \{y^h\} \) such that,
\[ \|y^{h+1} - y^\star\| \leq \|y^h - y^\star\| \]

**Proof.** From (25), one has that
\[ \|y^{h+1} - y^\star\|^2 = \|y^h - y^\star - \frac{1}{L} \nabla_y \Phi(y^h)\|^2 = \]
\[ = \|y^k - y^\star\|^2 + \frac{1}{L^2} \|\nabla_y \Phi(y^h)\|^2 - \frac{2}{L} \nabla_y \Phi(y^h) \nabla_y \Phi(y^h)^\top (y^h - y^\star). \]

Finally, substituting (28) in (27) we get
\[ \|y^{h+1} - y^\star\|^2 \leq \|y^k - y^\star\|^2 - \frac{1}{L^2} \|\nabla_y \Phi(y^h)\|^2 \]
which concludes the proof.

**Lemma 12.** Consider a sequence \( \{y^h\} \) generated as \( y^{h+1} = y^h - \frac{1}{L} \nabla \Phi(y^h) \) with \( y^0 \in Y \).

Then:

1. for all \( h \geq \bar{h} \) it holds that
   \[ \|\nabla \Phi(y^h)\| \leq L\|y^h - y^\star\| \]

2. if for some \( \bar{h} \) it holds that \( \|\nabla \Phi(y^\bar{h})\| = \varepsilon \), then
   \[ \|\nabla \Phi(y^h)\| \leq \frac{L \varepsilon}{\sigma} \]

   for all \( h \geq \bar{h} \).
Figure 5: Representation of the results of Lemma 12.

Proof. Using the right side of (24) and Lemma 11 one has that
\[ \| \nabla \Phi(y^{h+1}) \| \leq L \| y^{h+1} - y^* \| \leq L \| y^h - y^* \| \]

By induction, (29) follows directly and this concludes the proof for point 1. For point (ii), since \( \| \nabla \Phi(y^h) \| = \epsilon \), from the left side of (24), one has that
\[ \sigma \| y^h - y^* \| \leq \epsilon \]

which can be rewritten as
\[ \| y^h - y^* \| \leq \frac{\epsilon}{\sigma} \]

Then, substituting (31) in the right side of (24), we obtain (30) which, from point 1 concludes the proof.

A graphical representation of the previous Lemma is given in Figure 5. The gradient of \( \Phi(y) \) is bounded by the dotted lines, as from (24). Moreover, from (26), given \( \| \nabla_y \Phi(y^h) \| \), it holds that \( \| \nabla_y \Phi(y^{h+1}) \| \) stays in the shaded region.

Finally, Lemma 9 is proved by noting that, from Lemma 12
\[ \| \nabla \Phi(y^h) \| \leq \frac{L_i \epsilon_i}{\sigma} \]

for all \( h \geq \hat{h} \), and hence
\[ \| \nabla \Phi(y^h) \| = \sqrt{\sum_{i=1}^{N} (\| \nabla_y \Phi(y^h) \|)^2}. \]