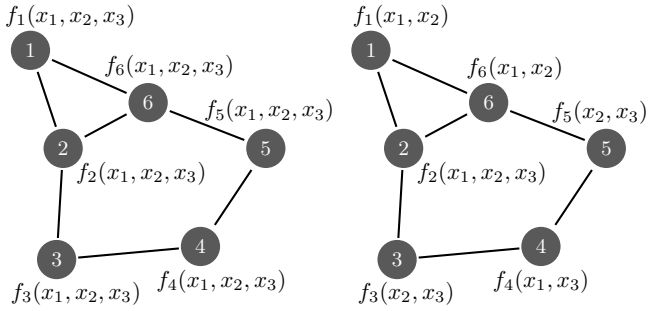
Distributed Optimization With Local Domains: Applications in MPC and Network Flows

João F. C. Mota, João M. F. Xavier, Pedro M. Q. Aguiar, and Markus Püschel

***Abstract*—In this paper we consider a network with P nodes, where each node has exclusive access to a local cost function. Our contribution is a communication-efficient distributed algorithm that finds a vector $x^\star$ minimizing the sum of all the functions. We make the additional assumption that the functions have intersecting local domains, i.e., each function depends only on some components of the variable. Consequently, each node is interested in knowing only some components of $x^\star$, not the entire vector. This allows for improvement in communication-efficiency. We apply our algorithm to model predictive control (MPC) and to network flow problems and show, through experiments on large networks, that our proposed algorithm requires less communications to converge than prior algorithms.**

***Index Terms*—Distributed algorithms, alternating direction method of multipliers (ADMM), Model Predictive Control, network flow, multicommodity flow, sensor networks.**



(a) Global variable (b) Partial variable

Figure 1. Example of a (a) global and a (b) partial variable. While each function in (a) depends on all the components of the variable $x = (x_1, x_2, x_3)$, each function in (b) depends only on a subset of the components of x.

I. Introduction

Distributed algorithms have become popular for solving optimization problems formulated on networks. Consider, for example, a network with P nodes and the following problem:

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f_1(x) + f_2(x) + \cdots + f_P(x)\,, \tag{1}$$

where f_p is a function known only at node p. Fig. 1(a) illustrates this problem for a variable x of size $n = 3$. Several algorithms have been proposed to solve (1) in a distributed way, that is, each node communicates only with its neighbors and there is no central node. In a typical distributed algorithm for (1), each node holds an estimate of a solution $x^\star$, and iteratively updates and exchanges it with its neighbors. It is usually assumed that all nodes are interested in knowing the entire solution $x^\star$. While such an assumption holds for problems like consensus [1] or distributed SVMs [2], there are important problems where it does not hold, especially in the context of large networks. Two examples we will explore here are distributed Model Predictive Control (MPC) and network flows. The goal in distributed MPC is to control a network of interacting subsystems with coupled dynamics [3]. That control should be performed using the least amount of energy. Network flow problems have many applications [4]; here, we will solve a network flow problem to minimize delays in a multicommodity routing problem. Both distributed MPC and network flow problems can be written naturally as (1) with functions that depend only on a subset of the components of x.

We solve (1) in the case that each function f_p may depend only on a subset of the components of the variable $x \in \mathbb{R}^n$. This situation is illustrated in Fig. 1(b), where, for example, f_1 only depends on x_1 and x_2. To capture these dependencies, we write x_S, $S \subseteq \{1, \ldots, n\}$, to denote a subset of the components of x. For example, if $S = \{2, 4\}$, then $x_S = (x_2, x_4)$. With this notation, our goal is solving

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f_1(x_{S_1}) + f_2(x_{S_2}) + \cdots + f_P(x_{S_P})\,, \tag{2}$$

where S_p is the set of components the function f_p depends on. Accordingly, every node p is only interested in a part of the solution: $x^\star_{S_p}$. We make the following

Assumption 1. *No components are global, i.e.,* $\bigcap_{p=1}^{P} S_p = \emptyset$.

Whenever this assumption holds, we say that the variable x is *partial*. Fig. 1(b) shows an example of a partial variable. Note that, although no component appears in all nodes, node 2 depends on all components, i.e., it has a global domain. In fact, Assumption 1 allows only a strict subset of nodes to have global domains. This contrasts with Fig. 1(a), where all nodes have global domains and hence Assumption 1 does not hold. We say that the variable x in Fig. 1(a) is *global*. Clearly, problem (2) is a particular case of problem (1) and hence it can be solved with any algorithm designed for (1). This approach, however, may introduce unnecessary communications, since nodes exchange full estimates of $x^\star$, and not just of the components they are interested in, thus

João F. C. Mota, João M. F. Xavier, and Pedro M. Q. Aguiar are with Instituto de Sistemas e Robótica (ISR), Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal.

Markus Püschel is with the Department of Computer Science at ETH Zurich, Switzerland.

João F. C. Mota is also with the Department of Electrical and Computer Engineering at Carnegie Mellon University, USA.

This work was supported by the following grants from Fundação para a Ciência e Tecnologia (FCT): CMU-PT/SIA/0026/2009, PEst-OE/EEI/LA0009/2011, and SFRH/BD/33520/2008 (through the Carnegie Mellon/Portugal Program managed by ICTI).

potentially wasting useful communication resources. In many networks, communication is the operation that consumes the most energy and/or time.

Contributions. We first formalize problem (2) by making a clear distinction between variable dependencies and communication network. Before, both were usually assumed the same. Then, we propose a distributed algorithm for problem (2) that takes advantage of its special structure to reduce communications. We will distinguish two cases for the variable of (2): connected and non-connected, and design algorithms for both. To our knowledge, this is the first time an algorithm has been proposed for a non-connected variable. We apply our algorithms to distributed MPC and to network flow problems. A surprising result is that, despite their generality, the proposed algorithms outperform prior algorithms even though they are application-specific.

Related work. Many algorithms have been proposed for the global problem (1), for example, gradient-based methods [1], [5], [6], or methods based on the *Alternating Direction Method of Multipliers* (ADMM) [7], [8], [9]. As mentioned before, solving (2) with an algorithm designed for (1) introduces unnecessary communications. We will observe this when we compare the algorithm proposed here with D-ADMM [9], the state-of-the-art for (1) in terms of communication-efficiency.

To our knowledge, this is the first time problem (2) has been explicitly stated in a distributed context. For example, [10, §7.2] proposes an algorithm for (2), but is not distributed in our sense. Namely, it either requires a platform that supports all-to-all communications (in other words, a central node), or requires running consensus algorithms on each induced subgraph, at each iteration [10, §10.1]. Thus, that algorithm is only distributed when every component induces subgraphs that are stars. Actually, we found only one algorithm in the literature that is distributed (or that can easily be made distributed) for all the scenarios considered in this paper. That algorithm was proposed in [11] in the context of power system state estimation (the algorithm we propose can also be applied to this problem, although we will not consider it here). Our simulations show that the algorithm in [11] requires always more communications than the algorithm we propose.

Although we found just one (communication-efficient) distributed algorithm solving (2), there are many other algorithms solving particular instances of it. For example, in network flow problems, each component of the variable is associated to an edge of the network. We will see such problems can be written as (2) with a connected variable, in the special case where each induced subgraph is a star. In this case, [10, §7.2] becomes distributed, and also gradient/subgradient methods can be applied directly either to the primal problem [12] or to the dual problem [13], and yield distributed algorithms. Network flow problems have also been tackled with Newton-like methods [14], [13]. A related problem is Network Utility Maximization (NUM), which is used to model traffic control on the Internet [15], [16]. For example, the TCP/IP protocol has been interpreted as a gradient algorithm solving a NUM. In [17], we compared a particular instance of the proposed algorithm with prior algorithms solving NUM, and showed that it requires less end-to-end communications. However, due to its structure, it does not offer interpretations of end-to-end protocols as realistic as gradient-based algorithms.

Distributed Model Predictive Control (MPC) [3] is another problem that has been addressed with algorithms solving (2), again in the special case of a variable whose components induce star subgraphs only. Such algorithms include subgradient methods [18], interior-point methods [19], fast gradient [20], and ADMM-based methods [20], [21] (which apply [10, §7.2]). All these methods were designed for the special case of star-shaped induced subgraphs and, similarly to [10, §7.2], they become inefficient if applied to more generic cases. In spite of its generality, the algorithm we propose requires less communications than previous algorithms that were specifically designed for distributed MPC or network flow problems.

Additionally, we apply our algorithm to two scenarios in distributed MPC that have not been considered before: problems where the variable is connected but the induced subgraphs are not stars, and problems with a non-connected variable. Both cases can model scenarios where subsystems that are coupled through their dynamics cannot communicate directly.

Lastly, this paper extends considerably our preliminary work [17]. In particular, the algorithm in [17] was designed for bipartite networks and was based on the 2-block ADMM. In contrast, the algorithms proposed here work on any connected network and are based on the Extended ADMM; thus, they have different convergence guarantees. Also, the MPC model proposed here is significantly more general than the one in [17].

II. Terminology and Problem Statement

We start by introducing the concepts of *communication network* and *variable connectivity*.

Communication network. A communication network is represented as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, P\}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. Two nodes communicate directly if there is an edge connecting them in $\mathcal{G}$. We assume:

Assumption 2. *$\mathcal{G}$ is connected and its topology does not change over time; also, a coloring scheme $\mathcal{C}$ of $\mathcal{G}$ is available beforehand.*

A coloring scheme $\mathcal{C}$ is a set of numbers, called colors, assigned to the nodes such that two neighbors never have the same color, as shown in Fig. 2. Given its importance in TDMA, a widespread protocol for avoiding packet collisions, there is a large literature on coloring networks, as briefly overviewed in [22]. Our algorithm integrates naturally with TDMA, since both use coloring as a synchronization scheme: nodes work sequentially according to their colors, and nodes with the same color work in parallel. The difference is that TDMA uses a more restrictive coloring, as nodes within two hops cannot have the same color. Note that packet collision is often ignored in the design of distributed algorithms, as confirmed by the ubiquitous assumption that all nodes can communicate simultaneously.

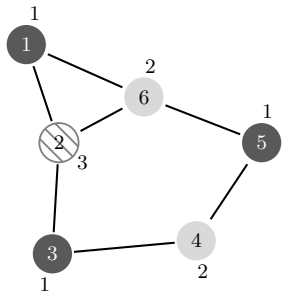


Figure 2. Example of a coloring scheme of the communication network using 3 colors: $\mathcal{C}_1 = \{1, 3, 5\}$, $\mathcal{C}_2 = \{4, 6\}$, and $\mathcal{C}_3 = \{2\}$.

We associate with each node p in the communication network a function $f_p : \mathbb{R}^{n_p} \to \mathbb{R} \cup \{+\infty\}$, where $n = n_1 + \cdots + n_P$, and make the

Assumption 3. *Each function f_p is known only at node p and it is closed, proper, and convex over $\mathbb{R}^{n_p}$.*

Since we allow f_p to take infinite values, each node can impose constraints on the variable using indicator functions, i.e., functions that evaluate to $+\infty$ when the constraints are not satisfied, and to 0 otherwise.

Variable connectivity. Although each function f_p is available only at node p, each component of the variable x may be associated with several nodes. Let x_l be a given component. The *subgraph induced by* x_l is represented by $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l) \subseteq \mathcal{G}$, where $\mathcal{V}_l$ is the set of nodes whose functions depend on x_l, and an edge $(i, j) \in \mathcal{E}$ belongs to $\mathcal{E}_l$ if both i and j are in $\mathcal{V}_l$. For example, the subgraph induced by x_1 in Fig. 1(b) consists of $\mathcal{V}_1 = \{1, 2, 4, 6\}$ and $\mathcal{E}_1 = \{(1, 2), (1, 6), (2, 6)\}$. We say that x_l is *connected* if its induced subgraph is connected, and *non-connected* otherwise. Likewise, a *variable is connected* if all its components are connected, and *non-connected* if it has at least one non-connected component.

Problem statement. Given a network satisfying Assumption 2 and a set of functions satisfying Assumptions 1 and 3, we solve the following problem: *design a distributed, communication-efficient algorithm that solves* (2)*, either with a connected or with a non-connected variable.*

By distributed algorithm we mean a procedure that makes no use of a central node and where each node communicates only with its neighbors. Unfortunately there is no known lower bound on how many communications are needed to solve (2). Because of this, communication-efficiency can only be assessed relative to existing algorithms that solve the same problem. As mentioned before, our strategy for this problem is to design an algorithm for the connected case and then generalize it to the non-connected case.

III. Connected Case

In this section we derive an algorithm for (2) assuming its variable is connected. Our derivation uses the same principles as the state-of-the-art algorithm [9], [22] for the global problem (1). The main idea is to manipulate (2) to make the Extended ADMM [23] applicable. We will see that the algorithm derived here generalizes the one in [9], [22].

Problem manipulation. Let x_l be a given component and $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ the respective induced subgraph. In this section we assume each $\mathcal{G}_l$ is connected. Since all nodes in $\mathcal{V}_l$ are interested in x_l, we will create a copy of x_l in each of those nodes: $x_l^{(p)}$ will be the copy at node p and $x_{S_p}^{(p)} := \{x_l^{(p)}\}_{l \in S_p}$ will be the set of all copies at node p. We rewrite (2) as

$$\begin{array}{ll} \underset{\{\bar{x}_l\}_{l=1}^n}{\text{minimize}} & f_1(x_{S_1}^{(1)}) + f_2(x_{S_2}^{(2)}) + \cdots + f_P(x_{S_P}^{(P)}) \\ \text{subject to} & x_l^{(i)} = x_l^{(j)}\,, \quad (i,j) \in \mathcal{E}_l\,,\ l = 1, \ldots, n\,, \end{array} \tag{3}$$

where $\{\bar{x}_l\}_{l=1}^L$ is the optimization variable and represents the set of all copies. We used $\bar{x}_l$ to denote all copies of the component x_l, which are located only in the nodes of $\mathcal{G}_l$: $\bar{x}_l := \{x_l^{(p)}\}_{p \in \mathcal{V}_l}$. The reason for introducing constraints in (3) is to enforce equality among the copies of the same component: if two neighboring nodes i and j depend on x_l, then $x_l^{(i)} = x_l^{(j)}$ appears in the constraints of (3). We assume that any edge in the communication network is represented as the ordered pair $(i, j) \in \mathcal{E}$, with $i < j$. As such, there are no repeated equations in (3). Problems (2) and (3) are equivalent because each induced subgraph is connected.

A useful observation is that $x_l^{(i)} = x_l^{(j)}$, $(i, j) \in \mathcal{E}_l$, can be written as $A_l \bar{x}_l = 0$, where A_l is the transposed node-arc incidence matrix of the subgraph $\mathcal{G}_l$. The node-arc incidence matrix represents a given graph with a matrix where each column corresponds to an edge $(i, j) \in \mathcal{E}$ and has 1 in the ith entry, -1 in the jth entry, and zeros elsewhere. We now partition the optimization variable according to the coloring scheme: for each $l = 1, \ldots, n$, $\bar{x}_l = (\bar{x}_l^1, \ldots, \bar{x}_l^C)$, where

$$\bar{x}_l^c = \begin{cases} \{x_l^{(p)}\}_{p \in \mathcal{V}_l \cap \mathcal{C}_c}\,, & \text{if } \mathcal{V}_l \cap \mathcal{C}_c \neq \emptyset \\ \emptyset\,, & \text{if } \mathcal{V}_l \cap \mathcal{C}_c = \emptyset \end{cases},$$

and $\mathcal{C}_c$ is the set of nodes that have color c. Thus, $\bar{x}_l^c$ is the set of copies of x_l held by the nodes that have color c. If no node with color c depends on x_l, then $\bar{x}_l^c$ is empty. A similar notation for the columns of the matrix A_l enables us to write $A_l \bar{x}_l$ as $\bar{A}_l^1 \bar{x}_l^1 + \cdots + \bar{A}_l^C \bar{x}_l^C$, and thus (3) equivalently as

$$\begin{array}{ll} \underset{\bar{x}^1, \ldots, \bar{x}^C}{\text{minimize}} & \sum_{p \in \mathcal{C}_1} f_p(x_{S_p}^{(p)}) + \cdots + \sum_{p \in \mathcal{C}_C} f_p(x_{S_p}^{(p)}) \\ \text{subject to} & \bar{A}^1 \bar{x}^1 + \cdots + \bar{A}^C \bar{x}^C = 0\,, \end{array} \tag{4}$$

where $\bar{x}^c = \{\bar{x}_l^c\}_{l=1}^n$, and $\bar{A}^c$ is the diagonal concatenation of the matrices $\bar{A}_1^c, \bar{A}_2^c, \ldots, \bar{A}_n^c$, i.e., $\bar{A}^c = \text{diag}(\bar{A}_1^c, \bar{A}_2^c, \ldots, \bar{A}_n^c)$. To better visualize the constraint in (4), we wrote

$$\bar{A}^c \bar{x}^c = \begin{bmatrix} \bar{A}_1^c & & & \\ & \bar{A}_2^c & & \\ & & \ddots & \\ & & & \bar{A}_n^c \end{bmatrix} \begin{bmatrix} \bar{x}_1^c \\ \bar{x}_2^c \\ \vdots \\ \bar{x}_n^c \end{bmatrix} \tag{5}$$

for each $c = 1, \ldots, C$. The format of (4) is exactly the one to which the Extended ADMM applies, as explained next.

Extended ADMM. The Extended ADMM is a natural generalization of the *Alternating Direction Method of Multipliers* (ADMM). Given a set of closed, convex functions $g_1, \ldots, g_C$, and a set of full column rank matrices $E_1, \ldots, E_C$, all with the same number of rows, the Extended ADMM solves

$$\begin{array}{ll} \underset{x_1, \ldots, x_C}{\text{minimize}} & g_1(x_1) + \cdots + g_C(x_C) \\ \text{subject to} & E_1 x_1 + \cdots + E_C x_C = 0\,. \end{array} \tag{6}$$

It consists of iterating on k the following equations:

$$x_1^{k+1} = \arg\min_{x_1} \; L_\rho(x_1, x_2^k, \ldots, x_P^k; \lambda^k) \tag{7}$$

$$x_2^{k+1} = \arg\min_{x_2} \; L_\rho(x_1^{k+1}, x_2, x_3^k, \ldots, x_C^k; \lambda^k) \tag{8}$$

$$\vdots$$

$$x_C^{k+1} = \arg\min_{x_C} \; L_\rho(x_1^{k+1}, x_2^{k+1}, \ldots, x_{C-1}^{k+1}, x_C; \lambda^k) \tag{9}$$

$$\lambda^{k+1} = \lambda^k + \rho \sum_{c=1}^{C} E_c x_c^{k+1}\,, \tag{10}$$

where λ is the dual variable, ρ is a positive parameter, and

$$L_\rho(x;\lambda) = \sum_{c=1}^{C} \big(g_c(x_c) + \lambda^\top E_c x_c\big) + \frac{\rho}{2}\Big\|\sum_{c=1}^{C} E_c x_c\Big\|^2$$

is the augmented Lagrangian of (6). The original ADMM is recovered whenever $C = 2$, i.e., when there are only two terms in the sums of (6). The following theorem gathers some known convergence results for (7)-(10).

Theorem 1 ([23], [24])**.** *For each $c = 1, \ldots, C$, let $g_c : \mathbb{R}^{n_c} \to \mathbb{R} \cup \{+\infty\}$ be closed and convex over $\mathbb{R}^{n_c}$ and $\operatorname{dom} g_c \neq \emptyset$. Let each E_c be an $m \times n_c$ matrix. Assume* (6) *is solvable and that either* 1) *$C = 2$ and each E_c has full column rank, or* 2) *$C \geq 2$ and each g_c is strongly convex. Then, the sequence $\{(x_1^k, \ldots, x_C^k, \lambda^k)\}$ generated by* (7)-(10) *converges to a primal-dual solution of* (6)*.*

It is believed that (7)-(10) converges even when $C > 2$, each g_c is closed and convex (not necessarily strongly convex), and each matrix E_c has full column rank. Such belief is supported by empirical evidence [22], [23] and its proof remains an open problem. So far, there are only proofs for modifications of (7)-(10) that resulted either in a slower algorithm [25], or in algorithms not applicable to distributed scenarios [26].

Applying the Extended ADMM. The clear correspondence between (4) and (6) makes (7)-(10) directly applicable to (4). Associate a dual variable λ_l^{ij} to each constraint $x_l^{(i)} = x_l^{(j)}$ in (3). Translating (10) component-wise, λ_l^{ij} is updated as

$$\lambda_l^{ij,k+1} = \lambda_l^{ij,k} + \rho\big(x_l^{(i),k+1} - x_l^{(j),k+1}\big)\,, \tag{11}$$

where $x_l^{(p),k+1}$ is the estimate of x_l at node p after iteration k. This estimate is obtained from (7)-(9), where we will focus our attention now. This sequence will yield the synchronization mentioned in Section II: nodes work sequentially according to their colors, with the same colored nodes working in parallel. In fact, each problem in (7)-(30) corresponds to a given color. Moreover, each of these problems decomposes into $|\mathcal{C}_c|$ problems that can be solved in parallel, each by a node with color c. For example, the copies of the nodes with color 1 are updated according to (7):

$$\bar{x}^{1,k+1} = \arg\min_{\bar{x}^1} \sum_{p\in\mathcal{C}_1} f_p(x_{S_p}^{(p)}) + {\lambda^k}^\top \bar{A}^1 \bar{x}^1 + \frac{\rho}{2}\Big\|\bar{A}^1\bar{x}^1 + \sum_{c=2}^{C} \bar{A}^c \bar{x}^{c,k}\Big\|^2 \tag{12}$$

$$= \arg\min_{\bar{x}^1} \sum_{p\in\mathcal{C}_1} \Bigg(f_p(x_{S_p}^{(p)}) + \sum_{l\in S_p} \sum_{j\in\mathcal{N}_p\cap\mathcal{V}_l} \Big(\operatorname{sign}(j-p)\lambda_l^{pj,k} - \rho\, x_l^{(j),k}\Big)^\top x_l^{(p)} + \frac{\rho}{2}\sum_{l\in S_p} D_{p,l}\Big(x_l^{(p)}\Big)^2\Bigg)\,, \tag{13}$$

whose equivalence is established in Lemma 1 below. In (13), the sign function is defined as 1 for nonnegative arguments and as -1 for negative arguments. Also, $D_{p,l}$ is the degree of node p in the subgraph $\mathcal{G}_l$, i.e., the number of neighbors of node p that also depend on x_l. Of course, $D_{p,l}$ is only defined when $l \in S_p$. Before establishing the equivalence between (12) and (13), note that (13) decomposes into $|\mathcal{C}_1|$ problems that can be solved in parallel. This is because $\bar{x}^1$ consists of the copies held by the nodes with color 1; and, since nodes with the same color are never neighbors, none of the copies in $\bar{x}^1$ appears as $x_l^{(j),k}$ in the second term of (13). Therefore, all nodes p in $\mathcal{C}_1$ can solve in parallel the following problem:

$$x_{S_p}^{(p),k+1} = \arg\min_{x_{S_p}^{(p)} = \{x_l^{(p)}\}_{l\in S_p}} f_p(x_{S_p}^{(p)}) + \sum_{l\in S_p} \sum_{j\in\mathcal{N}_p\cap\mathcal{V}_l} \Big(\operatorname{sign}(j-p)\lambda_l^{pj,k} - \rho\, x_l^{(j),k}\Big)^\top x_l^{(p)} + \frac{\rho}{2}\sum_{l\in S_p} D_{p,l}\Big(x_l^{(p)}\Big)^2. \tag{14}$$

However, node p can solve (14) only if it knows $x_l^{(j),k}$ and $\lambda_l^{pj,k}$, for $j \in \mathcal{N}_p \cap \mathcal{V}_l$ and $l \in S_p$. This is possible if, in the previous iteration, it received the respective copies of x_l from its neighbors. This is also enough for knowing $\lambda_l^{pj,k}$, although we will see later that no node needs to know each $\lambda_l^{pj,k}$ individually. The proof of the following lemma, in Appendix A, shows how we obtained (13) from (12).

Lemma 1. (12) *and* (13) *are equivalent.*

We just saw how (7) yields $|\mathcal{C}_1|$ problems with the format of (14) that can be solved in parallel by all the nodes with color 1. For the other colors, the analysis is the same with one minor difference: in the second term of (14) we have $x_l^{(j),k+1}$ from the neighbors with a smaller color and $x_l^{(j),k}$ from the nodes with a larger color.

The resulting algorithm is shown in Algorithm 1. There is a clear correspondence between the structure of Algorithm 1 and equations (7)-(10): steps 2-9 correspond to (7)-(9), and the loop in step 10 corresponds to (10). In steps 2-9, nodes work according to their colors, with the same colored nodes working in parallel. Each node computes the vector v in step 4, solves the optimization problem in step 6, and then sends the

Algorithm 1 Algorithm for a connected variable

Initialization: for all $p \in \mathcal{V}$, $l \in S_p$, set $\gamma_l^{(p),1} = x_l^{(p),1} = 0$; $k = 1$

1: **repeat**
2: **for** $c = 1, \ldots, C$ **do**
3: **for all** $p \in \mathcal{C}_c$ [in parallel] **do**
4: **for all** $l \in S_p$ **do**

$$v_l^{(p),k} = \gamma_l^{(p),k} - \rho \sum_{\substack{j \in \mathcal{N}_p \cap \mathcal{V}_l \\ C(j) < c}} x_l^{(j),k+1} - \rho \sum_{\substack{j \in \mathcal{N}_p \cap \mathcal{V}_l \\ C(j) > c}} x_l^{(j),k}$$

5: **end for**

6: Set $x_{S_p}^{(p),k+1}$ as the solution of

$$\underset{x_{S_p}^{(p)} = \{x_l^{(p)}\}_{l \in S_p}}{\arg\min} \; f_p(x_{S_p}^{(p)}) + \sum_{l \in S_p} v_l^{(p),k\top} x_l^{(p)} + \frac{\rho}{2} \sum_{l \in S_p} D_{p,l} \left(x_l^{(p)} \right)^2$$

7: For each component $l \in S_p$, send $x_l^{(p),k+1}$ to $\mathcal{N}_p \cap \mathcal{V}_l$

8: **end for**
9: **end for**
10: **for all** $p \in \mathcal{V}$ and $l \in S_p$ [in parallel] **do**

$$\gamma_l^{(p),k+1} = \gamma_l^{(p),k} + \rho \textstyle\sum_{j \in \mathcal{N}_p \cap \mathcal{V}_l} (x_l^{(p),k+1} - x_l^{(j),k+1})$$

11: **end for**
12: $k \leftarrow k + 1$
13: **until** some stopping criterion is met

new estimates of x_l to the neighbors that also depend on x_l, for $l \in S_p$. Note the introduction of extra notation in step 4: $C(p)$ is the color of node p. The computation of $v_l^{(p),k}$ in that step requires $x_l^{(j),k}$ from the neighbors with larger colors and $x_l^{(j),k+1}$ from the neighbors with smaller colors. While the former is obtained from the previous iteration, the latter is obtained at the current iteration, after the respective nodes execute step 7. Regarding the problem in step 6, it involves the private function of node p, f_p, to which is added a linear and a quadratic term. This fulfills our requirement that all operations involving f_p be performed at node p.

Note that the update of the dual variables in step 10 is different from (11). In particular, all the λ's at node p were condensed into a single dual variable $\gamma^{(p)}$. This was done because the optimization problem (14) does not depend on the individual λ_l^{pj}'s, but only on $\gamma_l^{(p),k} := \sum_{j \in \mathcal{N}_p \cap \mathcal{V}_l} \text{sign}(j-p) \lambda_l^{pj,k}$. If we replace

$$\lambda_l^{ij,k+1} = \lambda_l^{ij,k} + \rho \; \text{sign}(j-i) \big(x_l^{(i),k+1} - x_l^{(j),k+1} \big) \quad (15)$$

in the definition of $\gamma_l^{(p),k}$, we obtain the update of step 10. The extra "sign" in (15) (w.r.t. (11)) was necessary to take into account the extension of the definition of the dual variable λ_l^{ij} for $i > j$ (see Appendix A).

Convergence. Apart from manipulations, Algorithm 1 results from the application of the Extended ADMM to problem (4). Consequently, the conclusions of Theorem 1 apply if we prove that (4) satisfies the conditions of that theorem.

Lemma 2. *Each matrix $\bar{A}^c$ in* (4) *has full column rank.*

Proof: Let c be any color in $\{1, 2, \ldots, C\}$. By definition, $\bar{A}^c = \text{diag}(\bar{A}_1^c, \bar{A}_2^c, \ldots, \bar{A}_n^c)$; therefore, we have to prove that each $\bar{A}_l^c$ has full column rank, for $l = 1, 2, \ldots, n$. Let then c and l be fixed. We are going to prove that $(\bar{A}_l^c)^\top \bar{A}_l^c$, a square matrix, has full rank, and therefore $\bar{A}_l^c$ has full column rank. Since $\bar{A}_l = \begin{bmatrix} \bar{A}_1^c & \bar{A}_2^c & \cdots & \bar{A}_n^c \end{bmatrix}$, $(\bar{A}_l^c)^\top \bar{A}_l^c$ corresponds to the lth block in the diagonal of the matrix $A_l^\top A_l$, the Laplacian matrix of the induced subgraph $\mathcal{G}_l$. By assumption, in this section all induced subgraphs are connected. This means each node in $\mathcal{G}_l$ has at least one neighbor also in $\mathcal{G}_l$ and hence each entry in the diagonal of $A_l^\top A_l$ is greater than zero.[1] The same happens to the entries in the diagonal of $(\bar{A}_l^c)^\top \bar{A}_l^c$. In fact, these are the only nonzero entries of $(\bar{A}_l^c)^\top \bar{A}_l^c$, as this is a diagonal matrix. This is because $(\bar{A}_l^c)^\top \bar{A}_l^c$ corresponds to the Laplacian entries of nodes that have the same color, which are never neighbors. Therefore, $(\bar{A}_l^c)^\top \bar{A}_l^c$ has full rank. ■

The following corollary, whose proof is omitted, is a straightforward consequence of Theorem 1 and Lemma 2.

Corollary 1. *Let Assumptions 1-3 hold and let the variable be connected. Let also one of the following conditions hold:*

1) *the network is bipartite, i.e., $C = 2$, or*
2) *each $\sum_{p \in \mathcal{C}_c} f_p(x_{S_p})$ is strongly convex, $c = 1, \ldots, C$.*

Then, the sequence $\{x_{S_p}^{(p),k}\}_{k=1}^\infty$ at node p, produced by Algorithm 1, converges to $x_{S_p}^\star$, where $x^\star$ solves (2)*.*

As stated before, it is believed that the Extended ADMM converges for $C > 2$ even when none of the g_c's is strongly convex (just closed and convex). However, it is required that each E_c has full column rank. This translates into the belief that Algorithm 1 converges for any network, provided each f_p is closed and convex and each matrix $\bar{A}^c$ in (4) has full column rank. The last condition is the content of Lemma 2.

Comparison with other algorithms. Algorithm 1 is a generalization of D-ADMM [22]: by violating Assumption 1 and making $S_p = \{1, \ldots, n\}$ for all p, the variable becomes global and Algorithm 1 becomes exactly D-ADMM. This is a generalization indeed, for Algorithm 1 cannot be obtained from D-ADMM. The above fact is not surprising since Algorithm 1 was derived using the same set of ideas as D-ADMM, but adapted to a partial variable. Each iteration of Algorithm 1 (resp. D-ADMM) involves communicating $\sum_{p=1}^P |S_p|$ (resp. nP) numbers. Under Assumption 1, $\sum_{p=1}^P |S_p| < nP$, and thus there is a clear per-iteration gain in solving (2) with Algorithm 1. Although Assumption 1 can be ignored in the sense that Algorithm 1 still works without it, we considered that assumption to make clear the type of problems addressed in this paper.

We mentioned before that the algorithm in [11] is the only one we found in the literature that efficiently solves (2) in the same scenarios as Algorithm 1. For comparison purposes, we show it as Algorithm 2. Algorithms 1 and 2 are very similar in format, although their derivations are considerably different. In particular, Algorithm 2 is derived from the 2-block ADMM and thus it has stronger convergence guarantees. Namely, it does not require the network to be bipartite nor

[1] We are implicitly excluding the pathological case where a component x_l appears in only one node, say node p; this would lead to a Laplacian matrix $A_l^\top A_l$ equal to 0. This case is easily addressed by redefining f_p, the function at node p, to $\tilde{f}_p(\cdot) = \inf_{x_l} f_p(\ldots, x_l, \ldots)$.

Algorithm 2 [11]

Initialization: for all $p \in \mathcal{V}$, $l \in S_p$, set $\gamma_l^{(p),1} = x_l^{(p),1} = 0$; $k = 1$

repeat
  for all $p \in \mathcal{V}$ [in parallel] **do**
    for all $l \in S_p$ **do**

$$v_l^{(p),k} = \gamma_l^{(p),k} - \frac{\rho}{2}\Big(D_{p,l}x_l^{(p),k} + \sum_{j \in \mathcal{N}_p \cap \mathcal{V}_l} x_l^{(j),k}\Big)$$

    end for

    Set $x_{S_p}^{(p),k+1}$ as the solution of

$$\underset{x_{S_p}^{(p)} = \{x_l^{(p)}\}_{l \in S_p}}{\arg\min} \; f_p(x_{S_p}^{(p)}) + \sum_{l \in S_p} v_l^{(p),k\top} x_l^{(p)} + \frac{\rho}{2} \sum_{l \in S_p} D_{p,l}\Big(x_l^{(p)}\Big)^2$$

    For each component $l \in S_p$, send $x_l^{(p),k+1}$ to $\mathcal{N}_p \cap \mathcal{V}_l$
  end for
  for all $p \in \mathcal{V}$ and $l \in S_p$ [in parallel] **do**

$$\gamma_l^{(p),k+1} = \gamma_l^{(p),k} + \tfrac{\rho}{2} \textstyle\sum_{j \in \mathcal{N}_p \cap \mathcal{V}_l} (x_l^{(p),k+1} - x_l^{(j),k+1})$$

  end for
  $k \leftarrow k + 1$
until some stopping criterion is met

any function to be strongly convex (cf. Corollary 1). Also, it does not require any coloring scheme and, instead, all nodes perform the same tasks in parallel. Note also that the updates of $v_l^{(p)}$ and $\gamma_l^{(p)}$ are different in both algorithms. In the same way that Algorithm 1 was derived using the techniques of D-ADMM, Algorithm 2 was derived using the techniques of [7]. And, as in the experimental results of [22], [9], we will observe in Section VI that Algorithm 1 always requires less communications than Algorithm 2. Next, we propose a modification to Algorithms 1 and 2 that makes them applicable to a non-connected variable.

IV. Non-connected Case

So far, we have assumed a connected variable in (2). In this section, the variable will be non-connected, i.e., it will have at least one component that induces a non-connected subgraph. In this case, problems (2) and (3) are no longer equivalent and, therefore, the derivations that follow do not apply. We propose a small trick to make these problems equivalent.

Let x_l be a component whose induced subgraph $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ is non-connected. Then, the constraint $x_l^{(i)} = x_l^{(j)}$, $(i,j) \in \mathcal{E}_l$, in (3) fails to enforce equality on all the copies of x_l. To overcome this, we propose creating a "virtual" path to connect the disconnected components of $\mathcal{G}_l$. This will allow the nodes in $\mathcal{G}_l$ to reach an agreement on an optimal value for x_l. Since our goal is to minimize communications, we would like to find the "shortest path" between these disconnected components, that is, to find an optimal *Steiner tree*.

Steiner tree problem. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph and let $\mathcal{R} \subseteq \mathcal{V}$ be a set of *required nodes*. A Steiner tree is any tree in $\mathcal{G}$ that contains the required nodes, i.e., it is an acyclic connected graph $(\mathcal{T}, \mathcal{F}) \subseteq \mathcal{G}$ such that $\mathcal{R} \subseteq \mathcal{T}$. The set of nodes in the tree that are not required are called *Steiner nodes*, and will be denoted with $\mathcal{S} := \mathcal{T} \backslash \mathcal{R}$. In the *Steiner tree problem*, each edge $(i,j) \in \mathcal{E}$ has a cost c_{ij} associated, and the goal is to find a Steiner tree whose edges have a minimal cost. This is exactly our problem if we make $c_{ij} = 1$ for all edges and $\mathcal{R} = \mathcal{V}_l$. The Steiner tree problem is illustrated in Fig. 3, where the required nodes are black and the Steiner nodes are striped. Unfortunately, computing optimal Steiner trees is NP-hard [27]. There are, however, many heuristic algorithms, some even with approximation guarantees. The Steiner tree problem can be formulated as [28]

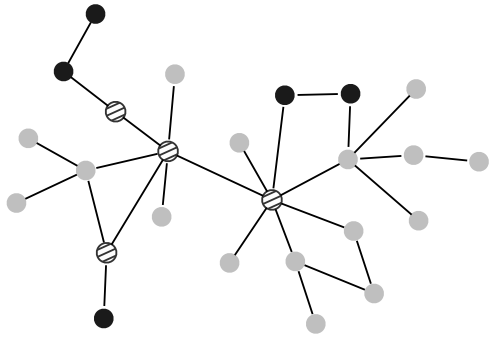


Figure 3. Example of an optimal Steiner tree: black nodes are required and striped nodes are Steiner.

$$\begin{aligned} \underset{\{z_{ij}\}_{(i,j)\in\mathcal{E}}}{\text{minimize}} \quad & \sum_{(i,j)\in\mathcal{E}} c_{ij} z_{ij} \\ \text{subject to} \quad & \sum_{\substack{i \in \mathcal{U} \\ j \notin \mathcal{U}}} z_{ij} \geq 1\,, \quad \forall \mathcal{U} \,:\, 0 < |\mathcal{U} \cap \mathcal{R}| < |\mathcal{R}| \\ & z_{ij} \in \{0,1\}\,, \quad (i,j) \in \mathcal{E}\,, \end{aligned} \tag{16}$$

where $\mathcal{U}$ in the first constraint is any subset of nodes that separates at least two required nodes. The optimization variable is constrained to be binary, and an optimal value $z_{ij}^\star = 1$ means that edge (i,j) was selected for the Steiner tree. Let $h(z) := \sum_{(i,j)\in\mathcal{E}} c_{ij} z_{ij}$ denote the objective of (16). We say that an algorithm for (16) has an approximation ratio of α if it produces a feasible point $\bar{z}$ such $h(\bar{z}) \leq \alpha h(z^\star)$, for any problem instance. For example, the primal-dual algorithm for combinatorial problems [28], [29] has an approximation ratio of 2. This number has been decreased in a series of works, the smallest one being $1 + \ln 3/2 \simeq 1.55$, provided by [30].

Algorithm generalization. To make Algorithms 1 and 2 applicable to a non-connected variable, we propose the following preprocessing step. For every component x_l that induces a disconnected subgraph $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$, compute a Steiner tree $(\mathcal{T}_l, \mathcal{F}_l) \subseteq \mathcal{G}$ using $\mathcal{V}_l$ as required nodes. Let $\mathcal{S}_l := \mathcal{T}_l \backslash \mathcal{V}_l$ denote the Steiner nodes in that tree. The functions of these Steiner nodes do not depend on x_l, i.e., $x_l \notin S_p$ for all $p \in \mathcal{S}_l$. Define a new induced graph as $\mathcal{G}_l' = (\mathcal{V}_l', \mathcal{E}_l')$, with $\mathcal{V}_l' := \mathcal{T}_l$ and $\mathcal{E}_l' := \mathcal{E}_l \cup \mathcal{F}_l$. Then, we can create copies of x_l in all nodes in $\mathcal{V}_l'$, and write (2) equivalently as

$$\begin{aligned} \underset{\{\bar{x}_l\}_{l=1}^n}{\text{minimize}} \quad & f_1(x_{S_1}^{(1)}) + f_2(x_{S_2}^{(2)}) + \cdots + f_P(x_{S_P}^{(P)}) \\ \text{subject to} \quad & x_l^{(i)} = x_l^{(j)}, \quad (i,j) \in \mathcal{E}_l'\,,\; l = 1, \ldots, n\,, \end{aligned} \tag{17}$$

where $\bar{x}_l := \{x_l^{(p)}\}_{p \in \mathcal{V}_l'}$ denotes the set of all copies of x_l, and $\{\bar{x}_l\}_{l=1}^L$, the optimization variable, represents the set of all copies. Note that the function at node p remains unchanged:

Algorithm 3 Algorithm for a non-connected variable

Preprocessing (centralized):

1: Set $S'_p = \emptyset$ for all $p \in \mathcal{V}$, and $\mathcal{V}'_l = \mathcal{V}_l$ for all $l = \{1, \dots, n\}$
2: **for all** $l \in \{1, \dots, n\}$ such that x_l is non-connected **do**
3: Compute a Steiner tree $(\mathcal{T}_l, \mathcal{F}_l)$, where $\mathcal{V}_l$ are required nodes
4: Set $\mathcal{V}'_l = \mathcal{T}_l$ and $\mathcal{S}_l := \mathcal{T}_l \backslash \mathcal{V}_l$ (Steiner nodes)
5: For all $p \in \mathcal{S}_l$, $S'_p = S'_p \cup \{x_l\}$
6: **end for**

Main algorithm (distributed):

Initialization: Set $\gamma_l^{(p),1} = x_l^{(p),1} = 0$, for $l \in S_p \cup S'_p$, $p \in \mathcal{V}$; $k = 1$
7: **repeat**
8: **for** $c = 1, \dots, C$ **do**
9: **for all** $p \in \mathcal{C}_c$ [in parallel] **do**
10: **for all** $l \in S_p \cup S'_p$ **do**

$$v_l^{(p),k} = \gamma_l^{(p),k} - \rho \sum_{\substack{j \in \mathcal{N}_p \cap \mathcal{V}'_l \\ C(j) < c}} x_l^{(j),k+1} - \rho \sum_{\substack{j \in \mathcal{N}_p \cap \mathcal{V}'_l \\ C(j) > c}} x_l^{(j),k}$$

11: **end for**
12: Set $x_{S_p \cup S'_p}^{(p),k+1}$ as the solution of

$$\arg\min_{x_{S_p \cup S'_p}^{(p)}} f_p(x_{S_p}^{(p)}) + \sum_{l \in S_p \cup S'_p} \left(v_l^{(p),k\top} x_l^{(p)} + \frac{\rho}{2} D_{p,l} \left(x_l^{(p)} \right)^2 \right)$$

13: For each $l \in S_p \cup S'_p$, send $x_l^{(p),k+1}$ to $\mathcal{N}_p \cap \mathcal{V}'_l$
14: **end for**
15: **end for**
16: **for all** $p \in \mathcal{V}$ and $l \in S_p \cup S'_p$ [in parallel] **do**

$$\gamma_l^{(p),k+1} = \gamma_l^{(p),k} + \rho \textstyle\sum_{j \in \mathcal{N}_p \cap \mathcal{V}'_l} (x_l^{(p),k+1} - x_l^{(j),k+1})$$

17: **end for**
18: $k \leftarrow k + 1$
19: **until** some stopping criterion is met

it only depends on $x_{S_p}^{(p)} := \{x_l^{(p)}\}_{l \in S_p}$, although node p can now have more copies, namely, $x_{S_p \cup S'_p}^{(p)}$, where S'_p is the set of components of which node p is a Steiner node. Of course, when a component x_l is connected, we set $\mathcal{G}'_l = \mathcal{G}_l$; also, if a node p is not Steiner for any component, $S'_p = \emptyset$. If we repeat the analysis of the previous section replacing problem (3) by (17), we get Algorithm 3.

Algorithm 3 has two parts: a preprocessing step, which is new, and the main algorithm, which it essentially Algorithm 1 with some small adaptations. We assume the preprocessing step can be done in a centralized way, before the execution of the main algorithm. In fact, the preprocessing only requires knowing the communication network $\mathcal{G}$ and the nodes' dependencies, but not the specific the functions f_p. Regarding the main algorithm, it is similar to Algorithm 1 except that each node, in addition to estimating the components its function depends on, it also estimates the components for which it is a Steiner node. The additional computations are, however, very simple: if node p is a Steiner node for component x_l, it updates it as $x_l^{(p),k+1} = -(1/(\rho\, D_{p,l}))v_l^{(p),k}$ in step 12; since f_p does not depend on x_l, the problem corresponding to the update of x_l becomes a quadratic problem for which there is a closed-form solution. Note that $D_{p,l}$ is now defined as the degree of node p in the subgraph $\mathcal{G}'_l$. The steps we took to generalize Algorithm 1 to a non-connected variable can be easily applied the same way to Algorithm 2.

V. Applications

In this section we describe how the proposed algorithms can be used to solve distributed MPC and network flow problems.

Distributed MPC. MPC is a popular control strategy for discrete-time systems [31]. It assumes a state-space model for the system, where the state at time t, here denoted with $x[t] \in \mathbb{R}^n$, evolves according to $x[t+1] = \Theta^t(x[t], u[t])$, where $u[t] \in \mathbb{R}^m$ is the input at time t and $\Theta^t : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a map that gives the system dynamics at each time instant t. Given a time-horizon T, an MPC implementation consists of measuring the state at time $t = 0$, computing the desired states and inputs for the next T time steps, applying $u[0]$ to the system, setting $t = 0$, and repeating the process. The second step, i.e., computing the desired states and inputs for a given time horizon T, is typically addressed by solving

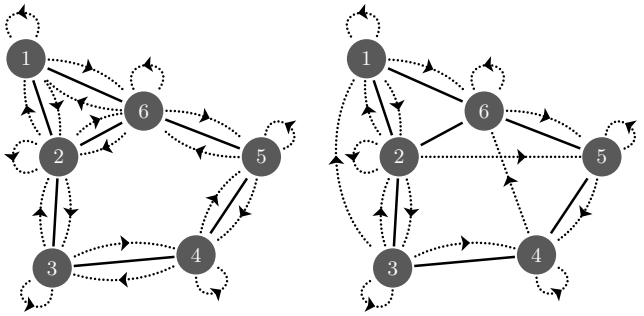
$$\begin{array}{ll} \underset{\bar{x},\bar{u}}{\text{minimize}} & \Phi(x[T]) + \sum_{t=0}^{T-1} \Psi^t(x[t], u[t]) \\ \text{subject to} & x[t+1] = \Theta^t(x[t], u[t]), \quad t = 0, \dots, T-1 \\ & x[0] = x^0, \end{array} \tag{18}$$

where the variable is $(\bar{x}, \bar{u}) := (\{x[t]\}_{t=0}^{T}, \{u[t]\}_{t=0}^{T-1})$. While Φ penalizes deviations of the final state $x[T]$ from our goal, Ψ^t usually measures, for each $t = 0, \dots, T-1$, some type of energy consumption that we want to minimize. Regarding the constraints of (18), the first one enforces the state to follow the system dynamics, and the second one encodes the initial measurement x^0.

We solve (18) in the following distributed scenario. There is a set of P systems that communicate through a communication network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each system has a state $x_p[t] \in \mathbb{R}^{n_p}$ and a local input $u_p[t] \in \mathbb{R}^{m_p}$, where $n_1 + \cdots + n_P = n$ and $m_1 + \cdots + m_P = m$. The state of system p evolves as $x_p[t+1] = \Theta_p^t\big(\{x_j[t], u_j[t]\}_{j \in \Omega_p}\big)$, where $\Omega_p \subseteq \mathcal{V}$ is the set of nodes whose state and/or input influences x_p (we assume $\{p\} \subseteq \Omega_p$ for all p). Note that, in contrast with what is usually assumed, Ω_p is not necessarily a subset of the neighbors of node p. In other words, two systems that influence each other may be unable to communicate directly. This is illustrated in Fig. 4(b) where, for example, the state/input of node 3 influences the state evolution of node 1 (dotted arrow), but there is no communication link (solid line) between them. Finally, we assume functions Φ and Ψ^t in (18) can be decomposed, respectively, as $\Phi(x[T]) = \sum_{p=1}^{P} \Phi_p(\{x_j[T]\}_{j \in \Omega_p})$ and $\Psi^t(x[t], u[t]) = \sum_{p=1}^{P} \Psi_p^t(\{x_j[t], u_j[t]\}_{j \in \Omega_p})$, where Φ_p and Ψ_p^t are both associated to node p. In sum, we solve

$$\begin{array}{ll} \underset{\bar{x},\bar{u}}{\min} & \sum_{p=1}^{P} \Big[\Phi_p(\{x_j[T]\}_{j \in \Omega_p}) \\ & \qquad + \sum_{t=0}^{T-1} \Psi_p^t(\{x_j[t], u_j[t]\}_{j \in \Omega_p}) \Big] \\ \text{s.t.} & x_p[t+1] = \Theta_p^t\big(\{x_j[t], u_j[t]\}_{j \in \Omega_p}\big), \ t = 0, \dots, T-1 \\ & x_p[0] = x_p^0 \\ & p = 1, \dots, P, \end{array} \tag{19}$$

where x_p^0 is the initial measurement at node p. The variable in (19) is $(\bar{x}, \bar{u}) := \big(\{\bar{x}_p\}_{p=1}^{P}, \{\bar{u}_p\}_{p=1}^{P}\big)$, where $\bar{x}_p :=$



(a) Connected star-shaped variable (b) Non-connected variable

Figure 4. Two MPC scenarios. Solid lines represent links in the communication network and dotted arrows represent system interactions. (a) Connected variable where each induced subgraph is a star. (b) Non-connected variable because node 5 is influenced by $(\bar{x}_2, \bar{u}_2)$, but not none of its neighbors are.

$\{x_p[t]\}_{t=0}^{T}$ and $\bar{u}_p := \{u_p[t]\}_{t=0}^{T-1}$. Problem (19) can be written as (2) by making

$$f_p(\{\bar{x}_j, \bar{u}_j\}_{j\in\Omega_p}) = \Phi_p(\{x_j[T]\}_{j\in\Omega_p}) + \mathrm{i}_{x_p[0]=x_p^0}(\bar{x}_p) + \sum_{t=0}^{T-1}\Big(\Psi_p^t(\{x_j[t], u_j[t]\}_{j\in\Omega_p}) + \mathrm{i}_{\Gamma_p^t}(\{\bar{x}_j, \bar{u}_j\}_{j\in\Omega_p})\Big),$$

where $\mathrm{i}_S(\cdot)$ is the indicator function of the set S, i.e., $\mathrm{i}_S(x) = +\infty$ if $x \notin S$ and $\mathrm{i}_S(x) = 0$ if $x \in S$, and $\Gamma_p^t := \{\{\bar{x}_j, \bar{u}_j\}_{j\in\Omega_p} : x_p[t+1] = \Theta_p^t(\{x_j[t], u_j[t]\}_{j\in\Omega_p})\}$.

We illustrate in Fig. 4(a) the case where $\Omega_p \subseteq \mathcal{N}_p \cup \{p\}$, i.e., the state of node p is influenced by its own state/input and by the states/inputs of the systems with which it can communicate. Using our terminology, this corresponds to a connected variable, where each induced subgraph is a star: the center of the star is node p, whose state is x_p. Particular cases of this model have been considered, for example, in [3], [32], [33], whose solutions are heuristics, and in [18], [19], [20], [21], whose solutions are optimization-based. The model we propose here is significantly more general, since it can handle scenarios where interacting nodes do not necessarily need to communicate, or even scenarios with a non-connected variable. Both cases are shown in Fig. 4(b). For example, the subgraph induced by $(\bar{x}_3, \bar{u}_3)$ consists of the nodes $\{1, 2, 3, 4\}$ and is connected. (The reference for connectivity is always the communication network which, in the plots, is represented by solid lines.) Nodes 1 and 3, however, cannot communicate directly. This is an example of an induced subgraph that is not a star. On the other hand, the subgraph induced by $(\bar{x}_2, \bar{u}_2)$ consists of the nodes $\{1, 2, 3, 5\}$. This subgraph is not connected, which implies that the optimization variable is non-connected. Situations like the above can be useful in scenarios where communications links are expensive or hard to establish. For instance, MPC can be used for temperature regulation of buildings [33], where making wired connections between rooms, here viewed as systems, can be expensive. In that case, two adjacent rooms whose temperatures influence each other may not be able to communicate directly. The proposed MPC model can handle this scenario easily.

MPC model for the experiments. We now present a simple linear MPC model, which will be used in our experiments in Section VI. Although simple, this model will illustrate all the cases considered above. We assume that systems are coupled though their inputs, i.e., $x_p[t+1] = A_p x_p[t] + \sum_{j\in\Omega_p} B_{pj} u_j[t]$, where $A_p \in \mathbb{R}^{n_p\times n_p}$ and each $B_{pj} \in \mathbb{R}^{n_p\times m_j}$ are arbitrary matrices, known only at node p. Also, we assume Φ_p and Ψ_p^t in (19) are, respectively, $\Phi_p(\{x_j[T]\}_{j\in\Omega_p}) = x_p[T]^\top \bar{Q}_p^f x_p[T]$ and $\Psi_p^t(\{x_j[t]\}_{j\in\Omega_p}) = x_p[t]^\top \bar{Q}_p x_p[t] + u_p[t]^\top \bar{R}_p$, where $\bar{Q}_p$ and $\bar{Q}_p^f$ are positive semidefinite matrices, and $\bar{R}_p$ is positive definite. Problem (19) then becomes

$$\begin{array}{ll}\underset{\substack{x_1,\ldots,x_P\\ u_1,\ldots,u_P}}{\text{minimize}} & \sum_{p=1}^{P} u_p^\top R_p u_p + x_p^\top Q_p x_p \\ \text{subject to} & x_p = C_p\{u_j\}_{j\in\mathcal{S}_p} + D_p^0\,,\ p = 1,\ldots,P\,,\end{array} \tag{20}$$

where, $x_p = (x_p[0], \ldots, x_p[T])$, $u_p = (u_p[0], \ldots, u_p[T-1])$, for each p, and

$$Q_p = \begin{bmatrix} I_T \otimes \bar{Q}_p & 0 \\ 0 & \bar{Q}_p^f \end{bmatrix}, \qquad R_p = I_T \otimes \bar{R}_p\,,$$

$$C_p = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ B_p & 0 & \cdots & 0 \\ A_{pp}B_p & B_p & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{pp}^{T-1}B_p & A_{pp}^{T-2}B_p & \cdots & B_p \end{bmatrix}, \quad D_p^0 = \begin{bmatrix} I \\ A_{pp} \\ A_{pp}^2 \\ \vdots \\ A_{pp}^T \end{bmatrix} x_p^0\,.$$

In the entries of matrix C_p, B_p is the horizontal concatenation of the matrices B_{pj}, for all $j \in \Omega_p$. One of the advantages of the model we are using is that all the variables x_p can be eliminated from (20), yielding

$$\underset{u_1,\ldots,u_P}{\text{minimize}} \ \sum_{p=1}^{P} \{u_j\}_{j\in S_p}^\top E_p \{u_j\}_{j\in S_p} + w_p^\top \{u_j\}_{j\in S_p}\,, \tag{21}$$

where each E_p is obtained by summing R_p with $C_p^\top Q_p C_p$ in the correct entries, and $w_p = 2C_p^\top Q_p D_p^0$. Our model thus leads to a very simple problem. In a centralized scenario, where all matrices E_p and all vectors w_p are known in the same location, the solution of (21) can be computed by solving a linear system. Likewise, the problem in step 6 of Algorithm 1 (and steps 5 and 12 of Algorithms 2 and 3, respectively) boils down to solving a linear system.

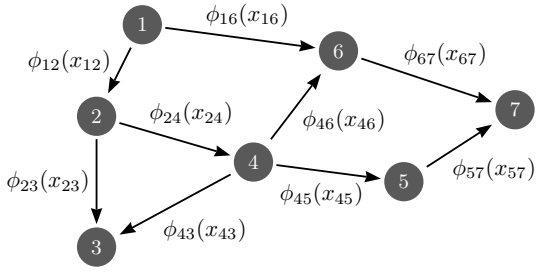


Figure 5. A network flow problem: each edge has a variable x_{ij} representing the flow from node i to node j and also has a cost function $\phi_{ij}(x_{ij})$.

Network flow. A network flow problem is typically formulated on a network with arcs (or directed edges), where an arc from node i to node j indicates a flow in that direction. In the example given in Fig. 5, there can be a flow from node 1 to node 6, but not the opposite. Every arc $(i, j) \in \mathcal{A}$ has associated a non-negative variable x_{ij} representing the

amount of flow in that arc (from node i to node j), and a cost function $\phi_{ij}(x_{ij})$ that depends only on x_{ij}. The goal is to minimize the sum of all the costs, while satisfying the laws of conservation of flow. External flow can be injected or extracted from a node, making that node a source or a sink, respectively. For example, in Fig. 5, node 1 can only be a source, since it has only outward edges; in contrast, nodes 3 and 7 can only be sinks, since they have only inward edges. The remaining nodes may or may not be sources or sinks. We represent the network of flows with the node-arc incidence matrix B, where the column associated to an arc from node i to node j has a -1 in the ith entry, a 1 in the jth entry, and zeros elsewhere. We assume the components of the variable x and the columns of B are in lexicographic order. For example, $x = (x_{12}, x_{16}, x_{23}, x_{24}, x_{43}, x_{45}, x_{46}, x_{57}, x_{67})$ would be the variable in Fig. 5. The laws of conservation of flow are expressed as $Bx = d$, where $d \in \mathbb{R}^P$ is the vector of external inputs/outputs. The entries of d sum up to zero and $d_p < 0$ (resp. $d_p > 0$) if node p is a source (resp. sink). When node p is neither a source nor a sink, $d_p = 0$. The problem we solve is

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sum_{(i,j)\in\mathcal{A}} \phi_{ij}(x_{ij}) \\ \text{subject to} & Bx = d \\ & x \geq 0\,, \end{array} \tag{22}$$

which can be written as (2) by setting

$$f_p\Big(\{x_{pj}\}_{(p,j)\in\mathcal{A}}, \{x_{jp}\}_{(j,p)\in\mathcal{A}}\Big) = \frac{1}{2}\sum_{(p,j)\in\mathcal{A}} \phi_{pj}(x_{pj})$$
$$+\frac{1}{2}\sum_{(j,p)\in\mathcal{A}} \phi_{jp}(x_{jp}) + \mathrm{i}_{b_p^\top x = d_p}(\{x_{pj}\}_{(p,j)\in\mathcal{A}}, \{x_{jp}\}_{(j,p)\in\mathcal{A}})\,,$$

where $b_p^\top$ is the pth row of B. In words, f_p consists of the sum of the functions associated to all arcs involving node p, plus the indicator function of the set $\{x : b_p^\top x = d_p\}$. This indicator function enforces the conservation of flow at node p and it only involves the variables $\{x_{pj}\}_{(p,j)\in\mathcal{A}}$ and $\{x_{jp}\}_{(j,p)\in\mathcal{A}}$.

Regarding the communication network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we assume it consists of the underlying undirected network. This means that nodes i and j can exchange messages directly, i.e., $(i,j) \in \mathcal{E}$ for $i < j$, if there is an arc between these nodes, i.e., $(i,j) \in \mathcal{A}$ or $(j,i) \in \mathcal{A}$. Therefore, in contrast with the flows, messages do not necessarily need to be exchanged satisfying the direction of the arcs. In fact, messages and flows might represent different physical quantities: think, for example, in a network of water pipes controlled by actuators at each pipe junction; while the pipes might enforce a direction in the flow of water (by using, for example, special valves), there is no reason to impose the same constraint on the electrical signals exchanged by the actuators. In problem (22), the subgraph induced by x_{ij}, $(i,j) \in \mathcal{A}$, consists only of nodes i and j and an edge connecting them. This makes the variable in (22) connected and star-shaped. Next we discuss the functions ϕ_{ij} used in our simulations.

Models for the experiments. We considered two instances of (22): a simple instance and a complex instance. While the simple instance makes all the algorithms we consider applicable, the (more) complex instance can be solved only by a subset of algorithms, but it provides a more realistic application. The simple instance uses $\phi_{ij} = \frac{1}{2}(x_{ij} - a_{ij})^2$, where $a_{ij} > 0$, as the cost function for each arc (i,j) and no constraints besides the conservation of flow, i.e., we drop the nonnegativity constraint $x \geq 0$ in (22). The reason for dropping this constraint was to make the algorithm in [13] applicable. The other instance we consider is [4, Ch.17]:

$$\begin{array}{ll} \underset{x=\{x_{ij}\}_{(i,j)\in\mathcal{A}}}{\text{minimize}} & \sum_{(i,j)\in\mathcal{A}} \frac{x_{ij}}{c_{ij}-x_{ij}} \\ \text{subject to} & Bx = d \\ & 0 \leq x_{ij} \leq c_{ij}\,, \quad (i,j) \in \mathcal{A}\,, \end{array} \tag{23}$$

where c_{ij} represents the capacity of the arc $(i,j) \in \mathcal{A}$. Problem (23) has the same format of (22) except for the additional capacity constraints $x_{ij} \leq c_{ij}$, and it models overall system delays on multicommodity flow problems [4, Ch.17]. If we apply Algorithm 1 to problem (23), node p has to solve at each step

$$\begin{array}{ll} \underset{y=(y_1,\ldots,y_{D_p})}{\text{minimize}} & \sum_{i=1}^{D_p}(\frac{y_i}{c_i-y_i} + v_i y_i + a_i y_i^2) \\ \text{subject to} & b_p^\top y = d_p \\ & 0 \leq y \leq c\,, \end{array} \tag{24}$$

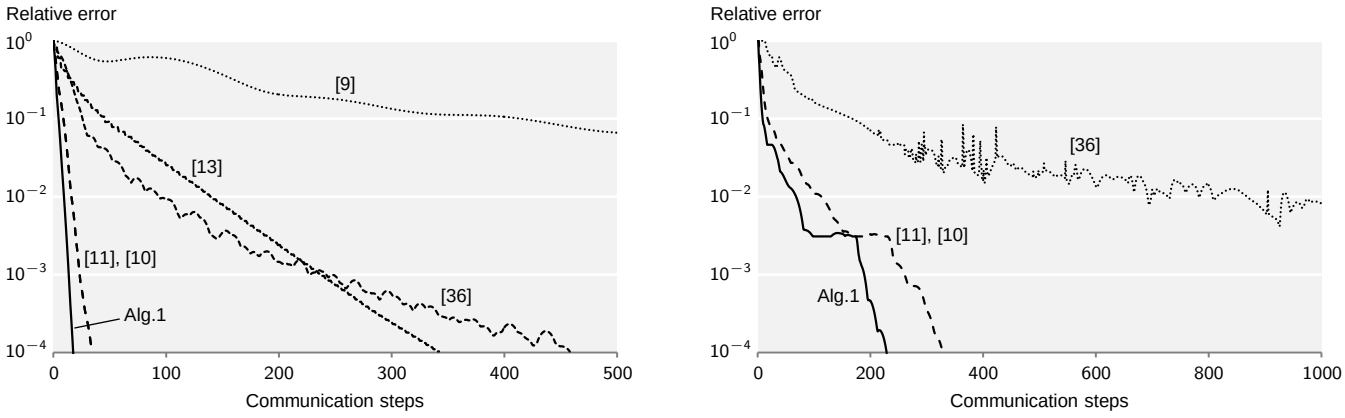
where each y_i corresponds to x_{pj} if $(p,j) \in \mathcal{A}$, or to x_{jp} if $(j,p) \in \mathcal{A}$. Since projecting a point onto the set of constraints of (24) is simple (see [34]), (24) can be solved efficiently with a projected gradient method. In fact, we will use the algorithm in [35], which is based on the Barzilai-Borwein method.

In sum, we will solve two instances of (22): a simple one, where $\phi_{ij}(x_{ij}) = (1/2)(x_{ij} - a_{ij})^2$ and with no constraints besides $Bx = d$, and (23), a more complex but realistic one.

VI. Experimental Results

In this section we show experimental results of the proposed algorithms solving MPC and network flow problems. We start with network flow because it is simpler and more algorithms are applicable. Also, it will illustrate the inefficiency of solving (2) with an algorithm designed for the global problem (1).

Network flows: experimental setup. As mentioned in the previous section, we solved two instances of (22). In both instances, we used a network with 2000 nodes and 3996 edges, generated randomly according to the Barabasi-Albert model [37] with parameter 2, using the Network X Python package [38]. We made the simplifying assumption that between any two pairs of nodes there can be at most arc, as shown in Fig. 5. Hence, the size of the variable x in (22) is equal to the number of edges $|\mathcal{E}|$, in this case, 3996. The generated network had a diameter of 8, an average node degree of 3.996, and it was colored with 3 colors in Sage [39]. This gives us the underlying (undirected) communication network. Then, we assigned randomly a direction to each edge, with equal probabilities for both directions, creating a directed network like in Fig. 5. We also assigned to each edge a number drawn randomly from the set $\{10, 20, 30, 40, 50, 100\}$. The probabilities were 0.2 for the first four elements and 0.1 for 50 and 100. These numbers played the role of the a_{ij}'s in



(a) Simple instance of (22)

(b) Problem (23)

Figure 6. Results for the network flow problems on a network with 2000 nodes and 3996 edges. The results in (a) are for the simple instance of (22), where $\phi_{ij}(x_{ij}) = (1/2)(x_{ij} - a_{ij})^2$ and there are no nonnegativity constraints; and the results in (b) are for (23).

the simple instance of (22) and the role of the capacities c_{ij} in (23). To generate the vector d or, in other words, to determine which nodes are sources or sinks, we proceeded as follows. For each $k = 1, \ldots, 100$, we picked a source s_k randomly (uniformly) out of the set of 2000 nodes and then picked a sink r_k randomly (uniformly) out of the set of reachable nodes of s_k. For example, if we were considering the network of Fig. 5 and picked $s_k = 4$ as a source node, the set of its reachable nodes would be $\{3, 5, 6, 7\}$. Next, we added to the entries s_k and r_k of d the values $-f_k/100$ and $f_k/100$, respectively, where f_k is a number drawn randomly exactly as c_{ij} (or a_{ij}). This corresponds to injecting a flow of quantity $f_k/100$ at node s_k and extracting the same quantity at node r_k. After repeating this process 100 times, for $k = 1, \ldots, K$, we obtained vector d.

To assess the error given by each algorithm, we computed the solutions $x^\star$ of the instances of (22) in a centralized way. The simple instance of (22) considers $\phi_{ij}(x_{ij}) = (1/2)(x_{ij} - a_{ij})^2$ and ignores the constraint $x \geq 0$. Thus, it is a simple quadratic program and has a closed-form solution: solving a linear system. Similarly, the problem Algorithm 1 (resp. Algorithm 2) has to solve in step 6 (resp. step 5) boils down to solving a linear system. To compute the solution of (23), the complex instance of (22), we used CVXOPT [40].

The plots we will show depict the relative error on the primal variable $\|x^k - x^\star\|_\infty / \|x^\star\|_\infty$, where x^k is the concatenation of the estimates at all nodes, versus the number of communication steps. A *communication step* (CS) consists of all nodes communicating their current estimates to their neighbors. That is, in each CS, information flows on each edge in both directions and, hence, the total number of CSs is proportional to the total number of communications. All the algorithms we compared, discussed next, had a tuning parameter: ρ for the ADMM-based algorithms (cf. Algorithms 1 and 2), a Lipschitz constant L for a gradient-based algorithm, and a stepsize α for a Newton-based algorithm. Suppose we selected $\bar{\rho}$ for an ADMM-based algorithm. We say that $\bar{\rho}$ *has precision* γ, if both $\bar{\rho} - \gamma$ and $\bar{\rho} + \gamma$ lead to worse results for that algorithm. A similar definition is used for L and α. We compared Algorithm 1, henceforth denoted as Alg. 1, against the ADMM-based algorithms in [10, §7.2] and [11] (recall that Algorithm 2 describes [11]), Nesterov's method [36], the distributed Newton method proposed in [13], and D-ADMM [9]. For network flow problems, the algorithms in [10, §7.2] and [11] coincide, i.e., they become exactly the same algorithm. This is not surprising since both are based on the same algorithm: the 2-block ADMM. All the ADMM-based algorithms, including Alg. 1, take 1 CS per iteration. The work in [13], besides proposing a distributed Newton method, also describes the application of the gradient method to the dual of (22). Here, instead of applying the simple gradient method, we apply Nesterov's method [36], which can be applied in the same conditions, has a better bound on the convergence rate, and is known to converge faster in practice. However, gradient methods, including Nesterov's method, require an objective that has a Lipschitz-continuous gradient. While this is the case of the objective of the dual of the simple instance of (22), the same does not happen for the objective of the dual of (23). Therefore, in the latter case, we had to estimate a Lipschitz constant L. Similarly to the ADMM-based algorithms, each iteration of a gradient algorithm takes 1 CS per iteration. Regarding the distributed Newton algorithm in [13], we implemented it with a parameter $N = 2$, which is the order of the approximation in the computation of the Newton direction, and fixed the stepsize α. With this implementation, each iteration takes 3 CSs. Finally, D-ADMM [9] is currently the most communication-efficient algorithm for the global problem (1). As such, it makes all the nodes compute the full solution $x^\star$, which has dimensions 3996 in this case. Thus, each message exchanged in one CS of D-ADMM is 3996 times larger than the messages exchanged by the other algorithms.

Network flows: results. The results for the simple instance of (22) are shown in Fig. 6(a). Of all the algorithms, Alg. 1 required the least amount of CSs to achieve any relative error between 1 and 10^{-4}. The second best were the algorithms [10] and [11], whose lines coincide because they become the same algorithm when applied to network flows. Nesterov's method [36] and the Newton-based method [13]

Table I
STATISTICS FOR THE NETWORKS USED IN MPC.

Name	Source	# Nodes	# Edges	Diam.	# Colors	Av. Deg.
A	[37]	100	196	6	3	3.92
B	[41]	4941	6594	46	6	2.67

had a performance very similar to each other, but worse than the ADMM-based algorithms. However, D-ADMM [9], which is also ADMM-based but solves the global problem (1) instead, was the algorithm with the worst performance. Note that, in addition to requiring much more CSs than any other algorithm, each message exchanged by [9] is 3996 times larger than a message exchanged by any other algorithm. This clearly shows that if we want to derive communication-efficient algorithms, we have to explore the structure of (1). Finally, we mention that the value of ρ in these experiments was 2 for all ADMM-based algorithms (precision 1), the Lipschitz constant L was 70 (precision 5), and the stepsize α was 0.4 (precision 0.1).

Fig. 6(b) shows the results for (23). In this case, we were not able to make the algorithm in [13] converge (actually, it is not guaranteed to converge for this problem). It is visible in Fig. 6(b) that this problem is harder to solve, since all algorithms required more CSs solve it. Again, Alg. 1 was the algorithm with the best performance. This time we did not find any choice for L that made Nesterov's algorithm [36] achieve an error of 10^{-4} in less than 1000 CSs. The best result we obtained was for $L = 15000$. The parameter ρ was 0.08 for Alg. 1 and 0.12 for [11], [10], both computed with precision 0.02.

MPC: experimental setup. For the MPC experiments we used two networks with very different sizes. One network, which we call A, has 100 nodes, 196 edges, and was generated the same way as the network for the network flow experiments: with a Barabasi-Albert model [37] with parameter 2. The other network, named B, has 4941 nodes and 6594 edges and it represents the topology of the Western States Power Grid [41] (obtained in [42]). The diameter, the number of used colors, and the average degree for these networks is shown in Table I. For coloring the networks, we used Sage [39].

We solved the MPC problem (21) and, to illustrate all the particular cases of a variable for (2), we created several types of data. For all the data types, the size of the state (resp. input) at each node was always $n_p = 3$ (resp. $m_p = 1$), and the time-horizon was $T = 5$. Since (21) has a variable of size m_pTP, network A implied a variable of size 500 and network B implied a variable of size 24705. With network A, we generated the matrices A_p so that each subsystem could be unstable; namely, we drew each of its entries from a normal distribution. With network B, we proceeded the same way, but then "shrunk" the eigenvalues of each A_p to the interval $[-1, 1]$, hence making each subsystem stable. All matrices B_{pj} were always generated as each A_p in the unstable case. The way we generated system couplings, i.e., the set Ω_p for each node p (see also the dotted arrows in the networks of Fig. 4), will be explained as we present the experimental results. Note that for the MPC problem (21) the Lipschitz constant of the gradient of its objective can be computed in closed-form and, therefore, does not need to be estimated. The relative error will be computed as in the network flows: $\|x^k - x^\star\|_\infty / \|x^\star\|_\infty$, where x^k is the concatenation of all the nodes' input estimates.

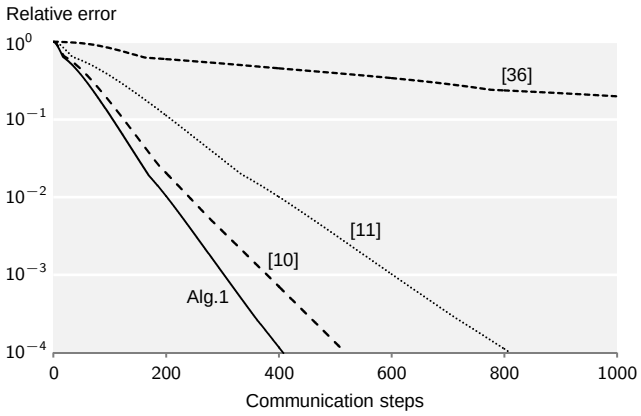
MPC results: connected case. The results for all the experiments on a connected variable are shown in Fig. 7. There, Alg. 1 is compared against [11] (see also Algorithm 2), and [10], and [36]. We mention that algorithms [10], [36] were already applied to (21), e.g., in [20], in the special case of a variable with star-shaped induced subgraphs. This is in fact the only case where [10] and [36] are distributed, and it explains why they are not in Figs. 7(c) and 7(d): the induced subgraphs in those figures are not stars. Only Alg. 1 and [11] are applicable in this case.

In Fig. 7(a) the network is A and each subsystem was generated (possibly) unstable, and in Fig. 7(b) the network is B and each subsystem was generated stable. In both cases, Alg. 1 required the least number of CSs to achieve any relative error between 1 and 10^{-4}, followed by [10], then by [11], and finally by [36]. It can be seen from these plots that the difficulty of the problem is determined, not so much by the size of network, but by the stability of the subsystems. In fact, all algorithms required uniformly more communications to solve a problem on network A, which has only 100 nodes, than on network B, which has approximately 5000 nodes. This difficulty can be measured by the Lipschitz constant L: 1.63×10^6 for network A (Fig. 7(a)) and 3395 for network B (Fig. 7(b)). Regarding the parameter ρ, in Fig. 7(a) it was 120 for [10] and 135 for the other algorithms (computed with precision 5); in Fig. 7(b), it was 25 for Alg. 1 and [10], and 30 for Alg. [11] (also computed with precision 5).

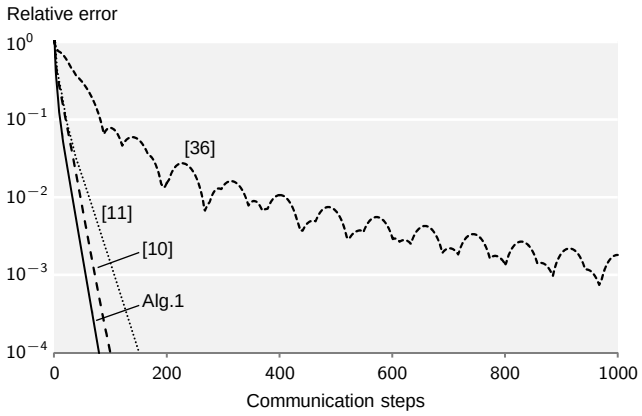
In Figs. 7(c) and 7(d) we considered a generic connected variable, where each induced subgraphs is not necessarily a star. In this case, the system couplings were generated as follows. Given a node p, we assigned it u_p and we initialized a fringe with its neighbors $\mathcal{N}_p$. Then, we selected a node randomly (with equal probability) from the fringe and made it depend on u_p; we also added its neighbors to the fringe. The described process was done 3 times for each variable u_p (i.e., node p). When each induced subgraph is not a star, only Alg. 1 and [11] are applicable. Figs. 7(c) and 7(d) show their performance for network A with unstable subsystems and for network B with stable subsystems, respectively. It can be seen that Alg. 1 required uniformly less CSs than [11] to achieve the same relative error.

MPC results: non-connected case. A non-connected variable has at least one component whose induced subgraph $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ is not connected. In this case, Algorithm 1 is no longer applicable and it requires a generalization, shown in Algorithm 3. Part of the generalization consists of computing Steiner trees, using the nodes in $\mathcal{V}_l$ as required nodes. The same generalization can be made to the algorithm in [11].

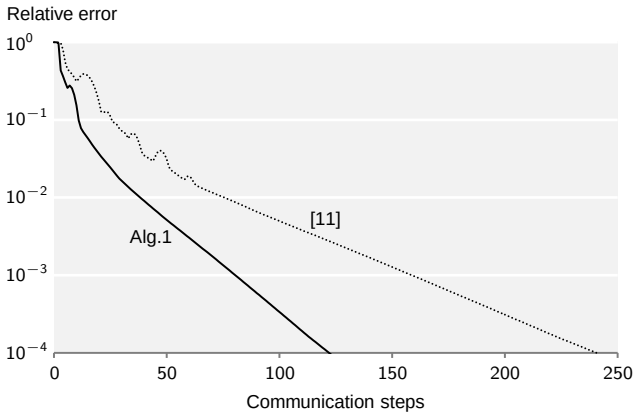
To create a problem instance with a non-connected variable, we generated system couplings in a way very similar to the couplings for Figs. 7(c) and 7(d). The difference was that any node in the network could be chosen to depend on a given u_p. However, any node in the fringe had twice the probability of



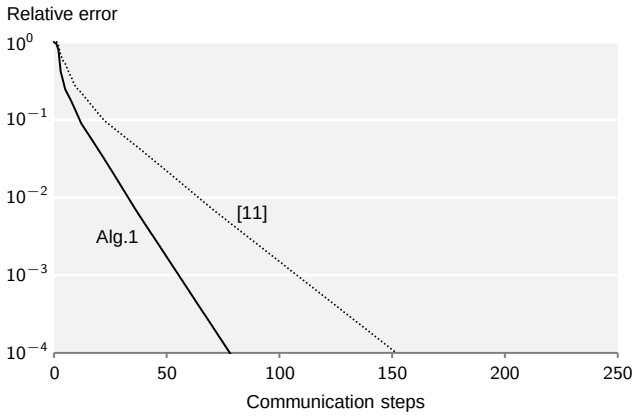
(a) Network A with star-shaped induced subgraphs



(b) Network B with star-shaped induced subgraphs



(c) Network A with a generic connected variable



(d) Network B with a generic connected variable

Figure 7. Results for MPC. The variable is connected in all cases, i.e., the subgraphs induced by all the components are connected. While in (a) and (b) each induced subgraph is a star, i.e., the interactions occur only between neighboring subsystems, in (c) and (d) each induced subgraph is generic. Only Algorithms 1 and 2 ([11]) are applicable in the latter case. Alg. 1 was always the algorithm requiring the least number of communication steps to converge.

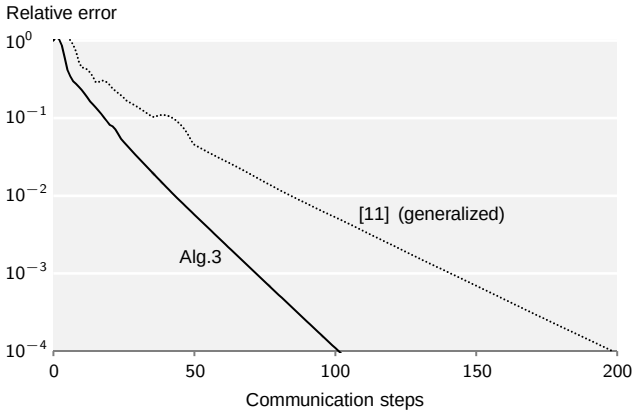


Figure 8. Results for MPC when the variable is non-connected. The communication network is A and all the subsytems were designed stable.

being chosen than any other node. This process was run on network A for each one of its 500 components (recall that the variable size for network A is 500), and obtained 400 non-connected components, i.e., 400 components whose induced subgraphs were not connected. Then, as described in the preprocessing part of Algorithm 3, we computed Steiner trees for each non-connected component: 44% of the nodes were Steiner for at least one component. To compute Steiner trees, we used a built-in Sage function [39]. In this case, we generated all the subsystems stable. Then, we ran Algorithms 3 and [11] (with a similar generalization) with $\rho = 35$ (computed with precision 5 for both algorithms). The results of these experiments are in Fig. 8. Again, Algorithm 3 required uniformly less CSs to converge than our generalization of [11].

VII. Conclusions

We solved a class of optimization problems with the following structure: no component of the optimization variable appears in the functions of all nodes. Our approach considers two different cases, a connected and a non-connected variable, and proposes an algorithm for each. Our algorithms require a coloring scheme of the network and their convergence is guaranteed only for the special case of a bipartite network or for problems with strongly convex objectives. However, in the practical examples that we considered, the algorithm converges even when none of these conditions is met. Moreover, experimental results show that our algorithms require less communications to solve a given network flow or MPC problem to an arbitrary level of accuracy than prior algorithms.

REFERENCES

[1] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proc. IPSN'04*, 2004, pp. 20–27.

[2] P. Forero, A. Cano, and G. Giannakis, "Consensus-based distributed support vector machines," *J. Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[3] E. Camponogara, D. Jia, B. Krogh, and S. Talukdar, "Distributed model predictive control," *IEEE Control Syst. Mag.*, vol. 22, no. 1, 2002.

[4] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.

[5] A. Nedić and Ozdaglar, *Convex Optimization in Signal Processing and Communications*, chapter Cooperative distributed multi-agent optimization, Cambridge University Press, 2010.

[6] D. Jakovetić, J. Xavier, and J. Moura, "Fast distributed gradient methods," http://arxiv.org/abs/1112.2972, 2011.

[7] H. Zhu, G. Giannakis, and A. Cano, "Distributed in-network channel decoding," *IEEE Trans. Sig. Proc.*, vol. 57, no. 10, 2009.

[8] I. Schizas, A. Ribeiro, and G. Giannakis, "Consensus in *ad hoc* wsns with noisy links - Part I: Distributed estimation of deterministic signals," *IEEE Trans. Sig. Proc.*, vol. 56, no. 1, pp. 350–364, 2008.

[9] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "D-ADMM: A communication-efficient distributed algorithm for separable optimization," to appear in IEEE Trans. Sig. Proc., 2013.

[10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, 2011.

[11] V. Kekatos and G. Giannakis, "Distributed robust power system state estimation," *IEEE Trans. Power Sys.*, vol. PP, no. 99, pp. 1–10, 2012.

[12] J. Tsitsiklis and D. Bertsekas, "Distributed asynchronous optimal routing in data networks," Tech. Rep., LIDS-P-1399, 1984.

[13] M. Zargham, A. Ribeiro, A. Jadbabaie, and A. Ozdaglar, "Accelerated dual descent for network optimization," http://arxiv.org/abs/1104.1157, 2012.

[14] D. Bertsekas and E. Gafni, "Projected Newton methods and optimization of multicommodity flows," *IEEE Trans. Aut. Contr.*, vol. AC-28, no. 12, 1983.

[15] F. Kelly, "Charging and rate control for elastic traffic," *Europ. Trans. Telecomm.*, vol. 8, pp. 33–37, 1997.

[16] S. Low, L. Peterson, and L. Wang, "Understanding Vegas: a duality model," *Journal of the ACM*, vol. 49, no. 2, 2002.

[17] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "Distributed ADMM for model predictive control and congestion control," in *51st IEEE Conf. Dec. Control*, 2012, pp. 5110–5115.

[18] Y. Wakasa, M. Arakawa, K. Tanaka, and T. Akashi, "Distributed model predictive control via dual decomposition," in *Conf. Dec. Control (CDC)*, 2008, pp. 381–386.

[19] E. Camponogara and H. Scherer, "Distributed optimization for model predictive control of linear dynamic networks with control-input and output constraints," *IEEE Trans. Aut. Sc. Engin.*, vol. 8, no. 1, 2011.

[20] C. Conte, T. Summers, M. Zeilinger, M. Morari, and C. Jones, "Computational aspects of distributed optimization in model predictive control," in *51st IEEE Conf. Decision and Contr. (CDC)*, 2012, pp. 6819–6824.

[21] T. Summers and J. Lygeros, "Distributed model predictive consensus via the alternating direction method of multipliers," in *50th Allerton Conf. Communication, Control, and Comp. IL, USA*, 2012.

[22] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "Distributed basis pursuit," *IEEE Trans. Sig. Proc.*, vol. 60, no. 4, 2012.

[23] D. Han and X. Yuan, "A note on the alternating direction method of multipliers," *J. Optim. Theory Appl.*, 2012.

[24] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "A proof of convergence for the alternating direction method of multipliers applied to polyhedral-constrained functions," http://arxiv.org/abs/1112.2295, 2011.

[25] M. Hong and Z. Luo, "On the linear convergence of the alternating direction method of multipliers," http://arxiv.org/abs/1208.3922, 2012.

[26] B. He, M. Tao, and X. Yuan, "Alternating direction method with Gaussian back substitution for separable convex programming," *SIAM J. Optim.*, vol. 22, no. 2, 2012.

[27] M. Garey and D. Johnson, "The rectilinear Steiner problem is NP-complete," *SIAM J. Appl. Math.*, vol. 32, pp. 826–834, 1977.

[28] D. Williamson, "The primal-dual method for approximating algorithms," *Math. Program.*, vol. 91, no. B, pp. 447–478, 2002.

[29] M. Goemans and D. Williamson, *Approximation algorithms for NP-hard problems*, chapter The primal-dual method for approximation algorithms and its application to network design problems, PWS Publishing Company, 1997.

[30] G. Robins and A Zelikovsky, "Improved Steiner tree approximation in graphs," in *11th anual ACM-SIAM symp. Discrete Algs.*, 2000, pp. 770–779.

[31] M. Morari and J. Lee, "Model predictive control: past, present and future," *Comp. and Chem. Eng.*, vol. 23, no. 4-5, pp. 667–682, 1999.

[32] T. Keviczky, F. Borrelli, and G. Balas, "Decentralized receding horizon control for large scale dynamically decoupled systems," *Automatica*, vol. 42, pp. 2105–2115, 2006.

[33] P. Moroşan, R. Bourdais, D. Dumur, and J. Buisson, "Distributed model predictive control for building temperature regulation," in *American Control Conf.*, 2010, pp. 3174–3179.

[34] L. Vandenberghe, "Optimization methods for large-scale systems," Spring 2008-09, Lecture Notes, UCLA.

[35] E. Birgin, J. Martinez, and M. Raydan, "Nonmonotone spectral projected gradient methods on convex sets," *SIAM J. Optim.*, vol. 10, no. 4, pp. 1196–1211, 2000.

[36] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Kluwer Academic Publishers, 2003.

[37] A. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[38] A. Hagberg, D. Schult, and P. Swart, "Exploring network structure, dynamics, and function using networkx," in *7th Python Sc. Conf., http://networkx.lanl.gov/index.html*, 2008.

[39] W. Stein et al., *Sage Mathematics Software*, The Sage Development Team, 2013, http://www.sagemath.org.

[40] M. Andersen, J. Dahl, and L. Vandenberghe, "CVXOPT," http://abel.ee.ucla.edu/cvxopt/index.html, 2012.

[41] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 409–10, 1998.

[42] M. Newman, "Power grid," http://www-personal.umich.edu/~Emejn/netdata/, retrieved on Feb 14, 2013.

APPENDIX A
PROOF OF LEMMA 1

To go from (12) to (13), we first develop the last two terms of (12), respectively,

$$\lambda^{k\top} \bar{A}^1 \bar{x}^1 \tag{25}$$

and

$$\frac{\rho}{2}\left\| \bar{A}^1 \bar{x}^1 + \sum_{c=2}^{C} \bar{A}^c \bar{x}^{c,k} \right\|^2. \tag{26}$$

We first address (25). Given the structure of $\bar{A}^1$, as seen in (5), we can write (25) as $\sum_{l=1}^{n}((\bar{A}_l^1)^\top \lambda_l^k)^\top \bar{x}_l^1$. Recall that $(\bar{A}_l^1)^\top$, if it exists (i.e., if there is a node with color 1 that depends on component x_l), consists of the block of rows of the node-arc incidence matrix of $\mathcal{G}_l$ corresponding to the nodes with color 1. Therefore, if there exists $p \in \mathcal{C}_1 \cap \mathcal{V}_l$, the vector $(\bar{A}_l^1)^\top \lambda_l^k$ will have an entry $\sum_{j \in \mathcal{N}_p \cap \mathcal{V}_l} \text{sign}(j-p)\lambda_l^{pj,k}$. The sign function appears here because the column of the node-arc incidence matrix corresponding to $x_l^{(i)} - x_l^{(j)} = 0$, for a pair $(i,j) \in \mathcal{E}_l$, contains 1 in the ith entry and -1 in the jth entry, where $i < j$. In the previous expression, we used an extension of the definition of λ_l^{ij}, which was only defined for $i < j$ (due to our convention that for any edge $(i,j) \in \mathcal{E}$ we have always $i < j$). Assume λ_l^{ij} is initialized with zero; switching i and j in (11), we obtain $\lambda_l^{ji,k} = -\lambda_l^{ij,k}$, which holds for all iterations k. To be consistent with the previous equation, we define λ_l^{ij} as

$\lambda_l^{ij} := -\lambda_l^{ji}$ whenever $i > j$. Therefore, (25) develops as

$$\begin{aligned}\lambda^{k\top}\bar{A}^1\bar{x}^1 &= \sum_{l=1}^{n}((\bar{A}_l^1)^\top\lambda_l^k)^\top\bar{x}_l^1 \\ &= \sum_{l=1}^{n}\sum_{p\in\mathcal{C}_1}\sum_{j\in\mathcal{N}_p\cap\mathcal{V}_l}\text{sign}(j-p)\left(\lambda_l^{pj,k}\right)^\top x_l^{(p)} \\ &= \sum_{p\in\mathcal{C}_1}\sum_{l=1}^{n}\sum_{j\in\mathcal{N}_p\cap\mathcal{V}_l}\text{sign}(j-p)\left(\lambda_l^{pj,k}\right)^\top x_l^{(p)}. \quad (27)\end{aligned}$$

Regarding (26), it can be written as

$$\begin{aligned}&\frac{\rho}{2}\left\|\bar{A}^1\bar{x}^1+\sum_{c=2}^{C}\bar{A}^c\bar{x}^{c,k}\right\|^2 \\ &= \frac{\rho}{2}\left\|\bar{A}^1\bar{x}^1\right\|^2+\rho(\bar{A}^1\bar{x}^1)^\top\sum_{c=2}^{C}\bar{A}^c\bar{x}^{c,k}+\frac{\rho}{2}\left\|\sum_{c=2}^{C}\bar{A}^c\bar{x}^{c,k}\right\|^2. \quad (28)\end{aligned}$$

Since the last term does not depend on $\bar{x}^1$, it can be dropped from the optimization problem. We now use the structure of $\bar{A}^1$ to rewrite the first term of (28):

$$\begin{aligned}\frac{\rho}{2}\left\|\bar{A}^1\bar{x}^1\right\|^2 &= \frac{\rho}{2}\sum_{l=1}^{n}(\bar{x}_l^1)^\top(\bar{A}_l^1)^\top\bar{A}_l^1\bar{x}_l^1 \quad (29)\\ &= \frac{\rho}{2}\sum_{l=1}^{n}\sum_{p\in\mathcal{C}_1}D_{p,l}\left(x_l^{(p)}\right)^2 \quad (30)\\ &= \frac{\rho}{2}\sum_{p\in\mathcal{C}_1}\sum_{l\in S_p}D_{p,l}\left(x_l^{(p)}\right)^2. \quad (31)\end{aligned}$$

From (29) to (30) we just used the structure of $\bar{A}_l^1$. Namely, if it exists, $(\bar{A}_l^1)^\top\bar{A}_l^1$ is a diagonal matrix, where each diagonal entry is extracted from the diagonal of $A_l^\top A_l$, the Laplacian matrix for $\mathcal{G}_l$. Since each entry in the diagonal of a Laplacian matrix contains the degrees of the respective nodes, the diagonal of $(\bar{A}_l^1)^\top\bar{A}_l^1$ contains $D_{p,l}$ for all $p\in\mathcal{C}_1$. The reason why $(\bar{A}_l^1)^\top\bar{A}_l^1$ is diagonal is because nodes with the same color are never neighbors. As in (28), we exchanged the order of the summations from (30) to (31).

Finally, we develop the second term of (28):

$$\begin{aligned}&\rho(\bar{A}^1\bar{x}^1)^\top\sum_{c=2}^{C}\bar{A}^c\bar{x}^{c,k} \\ &= \rho\sum_{c=2}^{C}\sum_{l=1}^{n}(\bar{x}_l^1)^\top(\bar{A}_l^1)^\top(\bar{A}_l^c)\,\bar{x}_l^{c,k} \quad (32)\\ &= -\rho\sum_{c=2}^{C}\sum_{l=1}^{n}\sum_{p\in\mathcal{C}_1}\sum_{j\in\mathcal{N}_p\cap\mathcal{C}_c\cap\mathcal{V}_l}x_l^{(p)\top}x_l^{(j),k} \quad (33)\\ &= -\rho\sum_{p\in\mathcal{C}_1}\sum_{l\in S_p}x_l^{(p)\top}\sum_{c=2}^{C}\sum_{j\in\mathcal{N}_p\cap\mathcal{C}_c\cap\mathcal{V}_l}x_l^{(j),k} \quad (34)\\ &= -\rho\sum_{p\in\mathcal{C}_1}\sum_{l\in S_p}\sum_{j\in\mathcal{N}_p\cap\mathcal{V}_l}x_l^{(p)\top}x_l^{(j),k}. \quad (35)\end{aligned}$$

In (32) we just used the structure of $\bar{A}^1$ and $\bar{A}^c$, as visualized in (5). From (32) to (33) we used the fact that $(\bar{A}_l^1)^\top\bar{A}_l^c$ is a submatrix of $A_l^\top A_l$, the Laplacian of $\mathcal{G}_l$, containing some of its off-diagonal elements. More concretely, $(\bar{A}_l^1)^\top\bar{A}_l^c$ contains the entries of $A_l^\top A_l$ corresponding to all the nodes $i\in\mathcal{C}_1\cap\mathcal{V}_l$ and $j\in\mathcal{C}_c\cap\mathcal{V}_l$. And, for such nodes, the corresponding entry in $A_l^\top A_l$ is -1 if i and j are neighbors, and 0 otherwise. From (34) to (35) we just used the fact that the set $\{\mathcal{C}_c\}_{c=2}^{C}$ is nothing but a partition of the set of neighbors of any node with color 1. Using (27), (28), (31), and (35) in (12), we get (13). □