Distributed ADMM for Model Predictive Control and Congestion Control

João F. C. Mota[1,2], João M. F. Xavier[2], Pedro M. Q. Aguiar[2], and Markus Püschel[3]

***Abstract*— Many problems in control can be modeled as an optimization problem over a network of nodes. Solving them with distributed algorithms provides advantages over centralized solutions, such as privacy and the ability to process data locally. In this paper, we solve optimization problems in networks where each node requires only partial knowledge of the problem's solution. We explore this feature to design a decentralized algorithm that allows a significant reduction in the total number of communications. Our algorithm is based on the Alternating Direction of Multipliers (ADMM), and we apply it to distributed Model Predictive Control (MPC) and TCP/IP congestion control. Simulation results show that the proposed algorithm requires less communications than previous work for the same solution accuracy.**

I. Introduction

Distributed processing and control techniques have been attracting considerable attention in several communities. New applications are requiring scalable distributed techniques due, for example, to the demand of processing massive amounts of data, or to the fact that data is generated in spatially different locations, as in sensor networks or the Internet.

In this paper we address distributed Model Predictive Control (MPC) and TCP/IP congestion control. Although both problems have been studied extensively (see [1], [2], [3], [4] for centralized and distributed MPC, and [5], [6] for congestion control), their distributed implementation still relies on classical techniques, for example, the gradient algorithm [2], [5]. On the other hand, in the optimization field, it has recently been shown that the Alternating Direction Method of Multipliers (ADMM) is more appropriate for distributed or parallel implementations [7], [8], [9].

The algorithm we propose here is decentralized and based on ADMM. It solves optimization problems with coupled cost functions in bipartite networks, requiring mild assumptions on those cost functions; for example, neither differentiability nor finite-valuedness is assumed, in contrast with the majority of the previous work. Furthermore, our simulations for distributed MPC and congestion control show that the proposed algorithm requires significantly less communications than state-of-the-art algorithms. Before formally stating the generic optimization problem that our algorithm solves, we introduce some notation.

*This work was supported by the following grants from Fundação para a Ciência e Tecnologia (FCT): CMU-PT/SIA/0026/2009, PTDC/EEA-ACR/73749/2006, PEst-OE/EEI/LA0009/2011, and SFRH/BD/33520/2008 (through the Carnegie Mellon/Portugal Program managed by ICTI).

[1]Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, USA

[2]Institute of Systems and Robotics, Instituto Superior Técnico, Technical University of Lisbon, Portugal

[3]Department of Computer Science, ETH Zurich, Switzerland

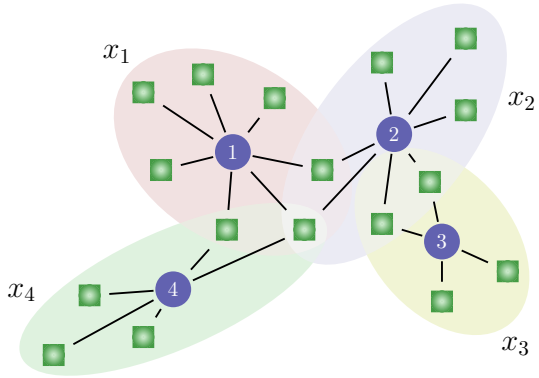


Fig. 1. Example bipartite graph with two types of nodes: central (circles) and peripheral (squares).

Notation. We assume a bipartite network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the set of nodes $\mathcal{V}$ is partitioned into two disjoint groups: *central nodes* $\mathcal{C} = \{1, \ldots, C\}$ and *peripheral nodes* $\mathcal{P} = \{1, \ldots, P\}$. The set of edges $\mathcal{E}$ connects the nodes in both groups: $\{i, j\} \in \mathcal{E}$ means that nodes i and j (belonging to different groups) are connected and thus can exchange information. Fig. 1 shows an example of a network where the nodes in $\mathcal{C}$ and $\mathcal{P}$ are represented with circles and squares, respectively. Given a central node c, we denote the set of its neighbors, which are all peripheral nodes, with $\mathcal{P}(c)$; similarly, we denote the set of neighbors of the peripheral node p with $\mathcal{C}(p)$.

Given a finite set $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ and a vector x_ω, indexed by a parameter $\omega \in \Omega$, let $\{x_\omega\}_{\omega \in \Omega}$ denote the n-tuple $(x_{\omega_1}, x_{\omega_2}, \ldots, x_{\omega_n})$. For simplicity, if $\Omega = \{\omega : A(\omega) \text{ holds}\}$, we represent $\{x_\omega\}_{\omega \in \Omega}$ with $\{x_\omega\}_{A(\omega) \text{ holds}}$.

Problem statement. Let $\mathcal{G}$ be a bipartite network as described above. All nodes are interested in solving an optimization problem with a global variable $x \in \mathbb{R}^n$. However, each node is interested in knowing only some components of x, not all of its entries. To model that, we partition the variable as $x = (x_1, x_2, \ldots, x_C)$, with $x_c \in \mathbb{R}^{n_c}$, $n_1 + \cdots + n_C = n$ and assume that each segment x_c is assigned to the central node $c \in \mathcal{C}$, which also has associated a cost function $f_c(x_c)$. Each peripheral node p, in turn, does not have a variable associated, but its cost function h_p depends on the variables of the central nodes to which it is connected, i.e., $h_p(\{x_c\}_{c \in \mathcal{C}(p)})$. We aim to solve

$$\underset{x=(x_1,\ldots,x_C)}{\text{minimize}} \quad \sum_{c \in \mathcal{C}} f_c(x_c) + \sum_{p \in \mathcal{P}} h_p(\{x_c\}_{c \in \mathcal{C}(p)}) \,. \qquad (1)$$

Any solution of (1) will be denoted with $x^\star = (x_1^\star, \dots, x_C^\star)$. Although problem (1) has no explicit constraints, these can be included by allowing each function to take infinite values. As Fig. 1 illustrates, the only nodes depending on x_c are $\{c\} \cup \mathcal{P}(c)$, i.e., the central node c and its neighbors. Our goal is: *given a bipartite network, design a distributed algorithm solving* (1) *such that only central node c and its neighbors exchange estimates of $x_c^\star$, $c = 1, \dots, C$.* By distributed algorithm, we mean that no node in the network except node c (resp. node p) has access to f_c (resp. h_p) at any time during or before the algorithm. We assume:

Assumption 1. *For $c = 1, \dots, C$, $f_c : \mathbb{R}^{n_c} \to \mathbb{R} \cup \{+\infty\}$ can be written as $f_c = \tilde{f}_c + i_{X_c}$, where $\tilde{f}_c$ is convex over $\mathbb{R}^{n_c}$ and i_{X_c} is the indicator function of a closed convex set X_c. Similarly, for $p = 1, \dots, P$, $h_p : \mathbb{R}^{m_p} \to \mathbb{R} \cup \{+\infty\}$, with $m_p = \sum_{c \in \mathcal{C}(p)} n_c$, can be written as $h_p = \tilde{h}_p + i_{X_p}$, with $\tilde{h}_p$ convex over $\mathbb{R}^{m_p}$ and X_p closed and convex.*

Assumption 2. *Problem* (1) *is solvable.*

Assumption 1 allows us to consider constraints in (1) through the use of indicator functions i_Ω: $\mathrm{i}_\Omega(\omega) = 0$ if $\omega \in \Omega$, and $\mathrm{i}_\Omega(\omega) = +\infty$ if $\omega \notin \Omega$. Assumption 2, however, requires all these constraints to have a nonempty intersection.

The question of how MPC and congestion control problems can be modeled as (1) will be addressed in section III.

Related work. Most of the literature that has tackled problems with the format (1) used gradient or subgradient algorithms. These can be applied directly to (1), as in [6], [10], or to a dual problem [11], [10]. Both cases require additional assumptions on the functions f_c and h_p. In particular, the gradient algorithm is applicable if the cost function is differentiable and its derivative Lipschitz continuous; otherwise, a subgradient algorithm has to be used, but it generally requires too many iterations to converge. Note that whenever it is possible to apply a gradient algorithm, its faster version (Nesterov's algorithm [12]) can also be applied. To the best of our knowledge, Nesterov's algorithm has never been studied for (1); nevertheless, we consider it for comparison with the proposed algorithm.

The work in [10], [11] applied the gradient algorithm to a dual problem of (1). Their original problem, however, contained an additional constraint coupling all the variables; as a consequence, their algorithm ran only on networks with a central node. If that constraint is absent, as here, their algorithm becomes decentralized. Unfortunately, this approach cannot be applied to solve the problems we consider here because it requires all the cost functions in (1) to be strictly convex, which is not the case in our applications; hence, no solution of (1) is recoverable just by solving its dual problem.

Another approach for solving (1) is with a distributed implementation of the barrier method with Newton's algorithm, yielding an algorithm with two nested loops; this approach was taken in [2] in the context of distributed MPC. Their simulations show that it slightly improves the convergence rate of the gradient method, both algorithms requiring about the same number of iterations.

Another line of related work has to do with

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \sum_{v \in \mathcal{V}} f_v(x) \\ \text{subject to} & x \in \bigcap_{v \in \mathcal{V}} X_v \,, \end{array} \tag{2}$$

where each function $f_v : \mathbb{R}^n \to \mathbb{R}$ and each set $X_v \subset \mathbb{R}^n$ are associated with the vth node of an arbitrary connected network. Problem (1) can be formulated as (2) and thus algorithms solving (2) also solve (1). So far, all the algorithms designed to solve (2) require the nodes to exchange full solution estimates among themselves, i.e., every communication consists of transmitting a vector with the same size as $x^\star$. Therefore, algorithms for (2) violate our requirement of exchanging only the necessary components. Nevertheless, we consider a state-of-the-art algorithm solving (2), [8], for comparison purposes; we will conclude that solving directly (1) with the algorithm we propose here reduces the number of communications significantly.

Finally, note that the algorithm proposed in [7, §7.2] cannot be applied to solve our problem, since it would require all-to-all communications in each step.

II. Proposed Algorithm

In this section we derive the algorithm. We start by manipulating (1) in order to make ADMM applicable.

Problem manipulations. Given a central node c, the only nodes depending on x_c are node c and its neighbors $\mathcal{P}(c)$. We thus replicate x_c throughout the nodes $\mathcal{P}(c)$, and denote the copy of node $p \in \mathcal{P}(c)$ with x_c^p. Problem (1) becomes

$$\begin{array}{ll} \underset{\bar{x} = (\bar{x}_1, \dots, \bar{x}_C)}{\text{minimize}} & \sum_{c \in \mathcal{C}} f_c(x_c) + \sum_{p \in \mathcal{P}} h_p(\{x_c^p\}_{c \in \mathcal{C}(p)}) \\ \text{subject to} & x_c = x_c^p \,, \quad p \in \mathcal{P}(c) \,, \; c = 1, \dots, C \,, \end{array} \tag{3}$$

where the new variable is $\bar{x} = (\bar{x}_1, \dots, \bar{x}_C)$ with $\bar{x}_c = (x_c, \{x_c^p\}_{p \in \mathcal{P}(c)})$. We now rewrite the constraint of (3) in a matrix format. We define the node-arc incidence matrix as a $P \times E$ matrix where each column is associated with an edge $\{i, j\}$ of the graph: if $i < j$, the ith (resp. jth) entry of that column is 1 (resp. -1), and vice-versa; the remaining entries are zeros. For each central node c, let A_c be the transpose of the node-arc incidence matrix of the subnetwork of nodes $\{c\} \cup \mathcal{P}(c)$: associate the $+1$ sign with the central node and the -1 sign with the peripheral nodes. Then, the constraints $x_c = x_c^p$, $p \in \mathcal{P}(c)$, can be written as $(A_c \otimes I_{n_c})\bar{x}_c = 0$, where I_q is the identity matrix in $\mathbb{R}^q$. Consequently, all the constraints in (3) can be written as $B\bar{x} = 0$, where B is the block diagonal matrix $\mathrm{Diag}(B_1, \dots, B_C)$ with $B_c = A_c \otimes I_{n_c}$.

Given a matrix $M \in \mathbb{R}^{m \times n}$ and a binary vector b in $\{0,1\}^n$, let M_b be the matrix obtained from removing the columns of M corresponding to the zero entries in b, i.e., M_b contains the ith column of M if $b_i = 1$. We adopt the same notation for vectors: x_b is the vector obtained from $x \in \mathbb{R}^n$ by removing the entries corresponding to $b_i = 0$. We define $v = (v_1, \dots, v_C)$, where $v_c \in \{0,1\}^{n_c(|\mathcal{P}(c)|+1)}$ has 1 in the first n_c entries and 0 in the remaining, for all c; let $u = (u_1, \dots, u_C)$, where $u_c \in \{0,1\}^{n_c(|\mathcal{P}(c)|+1)}$, be the complementary vector of v, i.e., u_c has a 0 in the

first n_c entries and ones in the remaining. There holds $B\bar{x} = B_v\bar{x}_v + B_u\bar{x}_u$, and hence (3) is written equivalently as

$$\begin{array}{ll}\underset{\bar{x}_v,\bar{x}_u}{\text{minimize}} & \sum_{c\in\mathcal{C}} f_c(x_c) + \sum_{p\in\mathcal{P}} h_p(\{x_c^p\}_{c\in\mathcal{C}(p)}) \\ \text{subject to} & B_v\bar{x}_v + B_u\bar{x}_u = 0\,. \end{array} \tag{4}$$

The terms $\sum_{c\in\mathcal{C}} f_c(x_c)$ and $\sum_{p\in\mathcal{P}} h_p(\{x_c^p\}_{c\in\mathcal{C}(p)})$ are functions of $\bar{x}_v$ and $\bar{x}_u$, respectively. Writing the problem in this format enables us to apply ADMM, described next.

ADMM. The Alternating Direction Method of Multipliers (ADMM) [7] is an efficient algorithm for solving

$$\begin{array}{ll}\underset{x_1\in X_1\,,\, x_2\in X_2}{\text{minimize}} & g_1(x_1) + g_2(x_2) \\ \text{subject to} & A_1x_1 + A_2x_2 = 0\,, \end{array} \tag{5}$$

where, for $i = 1, 2$, $g_i : \mathbb{R}^{n_i} \to \mathbb{R}$ is a convex function, X_i is a closed convex set, and A_i is a full column rank matrix. ADMM consists of iterating

$$x_1^{k+1} \in \arg\min_{x_1\in X_1} L_\rho(x_1, x_2^k; \lambda^k) \tag{6}$$

$$x_2^{k+1} \in \arg\min_{x_2\in X_2} L_\rho(x_1^{k+1}, x_2; \lambda^k) \tag{7}$$

$$\lambda^{k+1} = \lambda^k + \rho(A_1x_1^{k+1} + A_2x_2^{k+1})\,, \tag{8}$$

where L_ρ is the augmented Lagrangian of (5),

$$L_\rho(x_1, x_2; \lambda) := g_1(x_1) + g_2(x_2) + \lambda^\top(A_1x_1 + A_2x_2) + \frac{\rho}{2}\|A_1x_1 + A_2x_2\|^2\,, \tag{9}$$

λ is the dual variable, and $\rho > 0$ is a constant. The following theorem guarantees the convergence of (6)-(8).

Theorem 1 ([7], [8])**.** *Assume $g_i : \mathbb{R}^{n_i} \to \mathbb{R}$ is a convex function over $\mathbb{R}^{n_i}$, $X_i \subset \mathbb{R}^{n_i}$ is a closed and convex set, and A_i is a full column-rank matrix, for $i = 1, 2$. Assume problem* (5) *is solvable. Then, the sequence $\{(x_1^k, x_2^k, \lambda^k)\}$ generated by* (6)-(8) *converges to $(x_1^\star, x_2^\star, \lambda^\star)$, where $(x_1^\star, x_2^\star)$ solves* (5) *and $\lambda^\star$ solves the dual problem of* (5)*.*

Applying ADMM. We apply ADMM to problem (4) considering $\bar{x}_v$ and $\bar{x}_u$ as variables. At iteration k, $\bar{x}_v^{k+1}$ is found by solving

$$\min_{\bar{x}_v} \sum_{c\in\mathcal{C}}\Big(f_c(x_c) + \sum_{p\in\mathcal{P}(c)} ({\lambda_{cp}^k}^\top x_c + \frac{\rho}{2}\|x_c - x_c^{p,k}\|^2)\Big)\,, \tag{10}$$

where λ_{cp} is the dual variable associated to the constraint $x_c = x_c^p$. We assume there is a copy of λ_{cp} both in central node c and in peripheral node p. Similarly, after the central nodes transmit x_c^{k+1} to their neighbors, $\bar{x}_u$ is updated with the solution of

$$\min_{\bar{x}_u} \sum_{p\in\mathcal{P}}\Big(h_p\big(\{x_c^p\}_{c\in\mathcal{C}(p)}\big) + \sum_{c\in\mathcal{C}(p)} (-{\lambda_{cp}^k}^\top x_c^p + \frac{\rho}{2}\|x_c^{k+1} - x_c^p\|^2)\Big)\,. \tag{11}$$

Next, the copies of the dual variables are updated as

$$\lambda_{cp}^{k+1} = \lambda_{cp}^k + \rho\Big(x_c^{k+1} - x_c^{p,k+1}\Big)\,, \quad p \in \mathcal{P}(c)\,,\ c \in \mathcal{C}\,. \tag{12}$$

Since none of the terms in the objective of (10) is coupled, this problem decomposes into C problems that can be solved in parallel. Namely, central node c solves

$$\min_{x_c} f_c(x_c) + \Big(\gamma_c^k - \rho\sum_{p\in\mathcal{P}(c)} x_c^{p,k}\Big)^\top x_c + \frac{\rho D_c}{2}\|x_c\|^2\,, \tag{13}$$

where $D_c := |\mathcal{P}(c)|$ and $\gamma_c^k = \sum_{p\in\mathcal{P}(c)} \lambda_{cp}^k$. Regarding the $\bar{x}_u$ update, problem (11) also decomposes into P parallel problems. Namely, peripheral node p solves

$$\min_{\{x_c^p\}_{c\in\mathcal{C}(p)}} h_p\Big(\{x_c^p\}_{c\in\mathcal{C}(p)}\Big) - \{\lambda_{cp}^k + \rho x_c^{k+1}\}_{c\in\mathcal{C}(p)}^\top \{x_c^p\}_{c\in\mathcal{C}(p)} + \frac{\rho}{2}\Big\|\{x_c^p\}_{c\in\mathcal{C}(p)}\Big\|^2\,. \tag{14}$$

Note that peripheral node p needs to know the individual components of the vector $\{\lambda_{cp}^k\}_{c\in\mathcal{C}(p)}$, whereas central node c only needs to know the sum $\gamma_c^k = \sum_{p\in\mathcal{P}(c)} \lambda_{cp}^k$. The consequence is that, while peripheral nodes have to keep track of the neighbors' variables λ_{cp}, which can be done without any additional communication, the central nodes update γ_c instead of the components of λ they depend on. In fact, replacing (12) into the definition of γ_c^k yields $\gamma_c^{k+1} = \gamma_c^k + \rho D_c x_c^{k+1} - \rho\sum_{p\in\mathcal{P}(c)} x_c^{p,k+1}$. We name the resulting algorithm, described in Algorithm 1, Cluster-ADMM (C-ADMM). Note that C-ADMM has different procedures for the central nodes and for the peripheral nodes because we simplified the dual variable updates at the central nodes (see step 4). Apart from this, the algorithm is very similar for both types of nodes. The nodes exchange between themselves only the components of the variable that are of interest for them, not full solution estimates. Nevertheless, through cooperation, they solve (1). Its convergence is guaranteed by:

Theorem 2. *Let Assumptions 1-2 hold, let the given graph be connected and bipartite, and consider problem* (4)*. Then, $\sum_{c\in\mathcal{C}} \tilde{f}_c$ and $\sum_{p\in\mathcal{P}} \tilde{h}_p$ are convex over the full space, the sets X_c, $c \in \mathcal{C}$, and X_p, $p \in \mathcal{P}$, are closed and convex, and the matrices B_v and B_u have full column rank. Furthermore, problem* (4) *is solvable.*

Proof. Except the full column rankness of B_v and B_u, all conclusions of Theorem 2 follow directly from Assumptions 1-2. To see why B_v and B_u have full column rank, first partition B_c as $\big[(B_c)_{v_c} \quad (B_c)_{u_c}\big] = \big[(A_c \otimes I_{n_c})_{v_c} \quad (A_c \otimes I_{n_c})_{u_c}\big]$, for each $c = 1, \ldots, C$. The matrix $(A_c \otimes I_{n_c})_{v_c}$ (resp. $(A_c \otimes I_{n_c})_{u_c}$) has full column rank because it corresponds to a partitioning of a node-arc incidence matrix of a connected (star) graph (note that the Kronecker product by the identity matrix does not affect the result because it preserves ranks). Since $B_v = \text{Diag}((B_1)_{v_1}, \ldots, (B_C)_{v_C})$ and $B_u = \text{Diag}((B_1)_{u_1}, \ldots, (B_C)_{u_C})$, the result follows. □

III. Applications

We now describe how distributed Model Predictive Control and TCP/IP congestion control can be formulated as (1).

Algorithm 1 C-ADMM

Algorithm for central node $c \in \mathcal{C}$:
Initialization: Set $\gamma_c^1 = x_c^1 = 0$ and $k = 1$
repeat
  Set $v_c^k = \gamma_c^k - \rho \sum_{p\in\mathcal{P}(c)} x_c^{p,k}$ and find

$$x_c^{k+1} = \operatorname*{argmin}_{x_c} f_c(x_c) + {v_c^k}^\top x_c + \frac{\rho D_c}{2}\|x_c\|^2$$

  Send x_c^{k+1} to $\mathcal{P}(c)$
  Update $\gamma_c^{k+1} = \gamma_c^k + \rho D_c x_c^{k+1} - \rho \sum_{p\in\mathcal{P}(c)} x_c^{p,k+1}$
  $k \leftarrow k+1$
until some stopping criterion is met

Algorithm for peripheral node $p \in \mathcal{P}$:
Initialization: set $\lambda_{cp}^1 = 0$ for $c \in \mathcal{C}(p)$ and $k = 1$
repeat
  Set $v_p^k = -\{\lambda_{cp}^k - \rho\, x_c^{k+1}\}_{c\in\mathcal{C}(p)}$ and find

$$\{x_c^p\}_{\{c\in\mathcal{C}(p)\}}^{k+1} = \operatorname*{argmin}_{x_p} h_p(x_p) + {v_p^k}^\top x_p + \frac{\rho}{2}\|x_p\|^2$$

  for all $\mathbf{c} \in \mathcal{C}(\mathbf{p})$ **do**
    Send component x_c^p of $\{x_c^p\}_{\{c\in\mathcal{C}(p)\}}^{k+1}$ to central node c
    Update $\lambda_{cp}^{k+1} = \lambda_{cp}^k + \rho\left(x_c^{k+1} - x_c^{p,k+1}\right)$
  end for
  $k \leftarrow k+1$
until some stopping criterion is met

Model Predictive Control. Consider Fig. 1 and view the central nodes $\mathcal{C}$ as actuators and the peripheral nodes $\mathcal{P}$ as dynamical systems. Such network can be represented by a matrix $E \in \mathbb{R}^{P\times C}$, where $E_{pc} = 1$ if actuator c is connected to system p, and 0 otherwise. The goal is to control a global process modeled by $s(t+1) = As(t) + Bx(t)$, where $s(t) \in \mathbb{R}^n$ and $x(t) \in \mathbb{R}^m$ are, respectively, the state vector and the input at time t, and $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are fixed matrices. We decompose the global state $s(t)$ as $(s_1(t), \ldots, s_P(t))$, where $s_p(t) \in \mathbb{R}^{n_p}$ is the state of system p. Of course, $n_1 + \cdots + n_P = n$. We also decompose the input $x(t)$ as $(x_1(t), \ldots, x_C(t))$, where $x_c(t) \in \mathbb{R}^{m_c}$ is associated with the cth actuator. Again, $m_1 + \cdots + m_C = m$. This model was considered in [13], where a distributed linear controller $u(t) = Kx(t)$ was designed to stabilize the system. The network topology was encoded by imposing on K the same nonzero pattern as $E^\top$. This means that actuators only affect neighboring systems. We will make the same assumption here, but instead of designing a linear controller, we will design a Model Predictive Control (MPC) scheme. We also assume decoupled dynamics, i.e., $A = \text{Diag}(A_1, \ldots, A_P)$. The MPC problem we solve is

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T-1}\Big(s(t)^\top Q s(t) + x(t)^\top R x(t)\Big) \\ & \qquad + s(T)^\top Q_f s(T) \\ \text{subject to} & s(t+1) = As(t) + Bx(t)\,, \quad t = 0, \ldots, T-1 \\ & s(0) = s^0\,, \end{array} \tag{15}$$

where $(s(0), \ldots, s(T)) \in (\mathbb{R}^n)^{T+1}$ and $(x(0), \ldots, x(T-1)) \in (\mathbb{R}^m)^T$ are the variables, T is the prediction horizon, and s^0 is the observed state at $t = 0$. Matrices Q and Q_f are positive semidefinite, R is positive definite, and we assume all these matrices to be block diagonal: $Q = \text{Diag}(Q_1, \ldots, Q_P)$, $Q_f = \text{Diag}(Q_{f1}, \ldots, Q_{fP})$, $R = \text{Diag}(R_1, \ldots, R_C)$. Our assumptions enable decomposing the constraints of (15) as

$$\begin{cases} s_p(t+1) = A_p s_p(t) + \sum_{c\in\mathcal{C}(p)} B_{pc} x_c(t)\,, \quad t = 0, \ldots, T-1 \\ s_p(0) = s_p^0\,, \end{cases}$$

for each $p = 1, \ldots, P$, where $B_{ij} \in \mathbb{R}^{n_i\times m_j}$ is the ijth block of the matrix B (corresponding to the influence that actuator j exerts on system i), $s_p(t) \in \mathbb{R}^{n_p}$ is the pth block of $s(t)$, and $s_p^0 \in \mathbb{R}^{n_p}$ is the pth block of s^0. Writing these constraints in matrix form and replacing them in the objective of (15) yields a problem with the format of (1):

$$\begin{aligned} \min \sum_{c\in\mathcal{C}} \bar{x}_c^\top \bar{R}_c \bar{x}_c + \sum_{p\in\mathcal{P}} \big(\{\bar{x}_c\}_{c\in\mathcal{C}(p)}^\top \bar{H}_p^\top \bar{Q}_p \bar{H}_p \{\bar{x}_c\}_{c\in\mathcal{C}(p)} \\ + 2 s_p^0 \bar{G}_p^\top \bar{Q}_p \bar{H}_p \{\bar{x}_c\}_{c\in\mathcal{C}(p)}\big)\,, \end{aligned} \tag{16}$$

with variable $(\bar{x}_1, \ldots, \bar{x}_C)$, where $\bar{x}_c = (x_c(0), \ldots, x_c(T-1)) \in \mathbb{R}^{m_c T}$. In (16), $\bar{R}_c = I_T \otimes R_c$, $\bar{Q}_p = \text{Diag}(I_T \otimes Q_p, Q_{f_p})$, $\bar{G}_p = \begin{bmatrix} I_{n_p} & A_p & A_p^2 & \cdots & A_p^T \end{bmatrix}^\top$, and

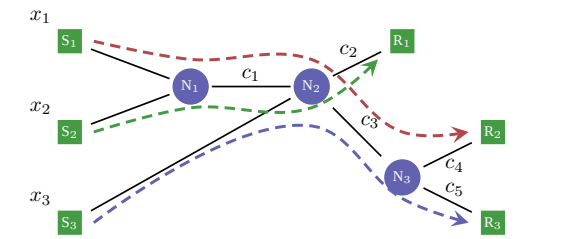
$$\bar{H}_p = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ B_p & 0 & \cdots & 0 \\ A_p B_p & B_p & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_p^{T-1} B_p & A_p^{T-2} B_p & \cdots & B_p \end{bmatrix}. \tag{17}$$

In (17), B_p is the horizontal concatenation of the matrices B_{pc} for $c \in \mathcal{C}(p)$. Having the format of (1), problem (16) can be solved either with C-ADMM or with gradient methods. While formulating (15) as (16) allows a distributed solution, it also makes the Lipschitz constant of the gradient of (16) large, due to (17). Consequently, gradient methods have difficulties handling problem (16). C-ADMM, on the other hand, has little sensitivity to Lipschitz constants and thus it solves (16) efficiently.

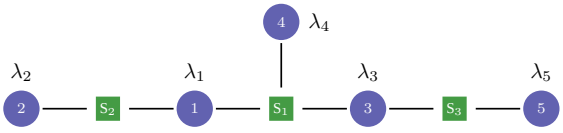
Other approaches for distributed MPC have been proposed before, but assume different models. For example, [4] proposes an algorithm for solving (15), where each node is simultaneously a system and an actuator; and [3] proposes an heuristic to solve (15) for coupled systems, assuming communication links between interdependent systems.

Congestion control. Congestion control is very important in networking since it prevents overloading networks with (lost) packets. Many congestion control protocols (e.g., TCP Vegas) have been modeled as a distributed solution, usually the gradient algorithm, for a utility maximization problem [5], [6]. We propose C-ADMM as an alternative.

Consider a network with three types of nodes: source nodes $\mathcal{S}$, recipient nodes $\mathcal{R}$, and intermediate nodes $\mathcal{N}$, as in Fig. 2. Each link l in the network has finite capacity $c_l > 0$ and, for simplicity, only supports flows in one direction. The case of flows in both directions can be easily generalized. Consider also a set of predetermined routes from the source



(a) Original network



(b) Bipartite graph obtained from (a)

Fig. 2. (a) network with $|\mathcal{S}| = |\mathcal{R}| = |\mathcal{N}| = 3$; each source transmits packets to a single recipient; (b) graph obtained from (a): each link from (a) with a capacity associated is represented as a circular node in (b).

nodes to the recipient nodes such that each source has only one recipient for its packets. Our goal is to compute the "optimal" sending rate x_s for each source $s \in \mathcal{S}$ such that the link capacity constraints are satisfied. For example, the TCP Vegas protocol was modeled as solving [5], [6]:

$$\begin{array}{ll} \underset{\{x_s\}_{s\in\mathcal{S}}}{\text{maximize}} & \sum_{s\in\mathcal{S}} w_s \log x_s \\ \text{subject to} & \sum_{s\in\mathcal{S}(l)} x_s \leq c_l\,, \quad l = 1,\ldots,L\,, \end{array} \tag{18}$$

where $w_s > 0$ is a parameter associated to source s, and $\mathcal{S}(l)$ is the set of sources that use link l. Since the objective of (18) is strictly concave, we can solve its dual problem instead:

$$\begin{array}{ll} \underset{\lambda=(\lambda_1,\ldots,\lambda_L)}{\text{minimize}} & \sum_{l\in\mathcal{L}} c_l\lambda_l - \sum_{s\in\mathcal{S}} w_s \log(\sum_{l\in\mathcal{L}(s)} \lambda_l) \\ \text{subject to} & \lambda_l \geq 0\,, \quad l \in \mathcal{L}\,, \end{array} \tag{19}$$

where $\mathcal{L}$ is the set of links, $L := |\mathcal{L}|$, and $\mathcal{L}(s)$ is the set of links that source s uses in its route. Once a solution $\lambda^\star$ of (19) is found, the optimal rate for source s can be found as $x_s^\star = w_s / \sum_{l\in\mathcal{L}(s)} \lambda_l^\star$. Problem (19) has the same format of (1) if we see each link as a central node and each source as a peripheral node. To make this connection clearer, we construct a bipartite graph from the original network the following way: if $l \in \mathcal{L}$, create a central node type with the label l, and let the source nodes $s \in \mathcal{S}$ be the peripheral nodes; if source node s uses link l, connect peripheral node s to central node l. Fig. 2(b) shows the bipartite graph obtained from 2(a) by considering only the links marked with capacities. Now the connection between (19) and (1) is clearer: each central node l holds λ_l and has a linear objective $f_l(\lambda_l) = c_l\lambda_l$; each peripheral node s has an objective that depends on the variables of the central nodes to which it is connected, i.e., $h_s(\{\lambda_l\}_{l\in\mathcal{L}(s)}) = w_s \log(\sum_{l\in\mathcal{L}(s)} \lambda_l)$. Therefore, C-ADMM can be applied and it provides a new protocol in the scope of the interpretation of [6].

According to that interpretation, who manages the dual variable λ_l, which can be seen as a price for using link l, is the intermediate node that has link l as an output. For example, in Fig. 2, node N_2 manages both λ_2 and λ_3. The communication between neighboring nodes in the bipartite graph of Fig. 2(b) is carried out along the routes drawn in Fig. 2(a), and it can be implicit (e.g., missing acknowledgment packets from intermediate node l can be seen as a raise in the price λ_l) or explicit (e.g., by adding specific fields to the packets for price negotiation). In a C-ADMM implementation, while the node managing link l would only require the aggregate sum of the rates of the sources using link l (step 8 of Algorithm 1), each source s would have to know the full vector $\{\lambda_{ls}\}_{l\in\mathcal{C}(s)}$. In a gradient method implementation [6], both the sources and the intermediate nodes only require knowledge of aggregate quantities. C-ADMM, however, provides faster convergence rates, as shown next.

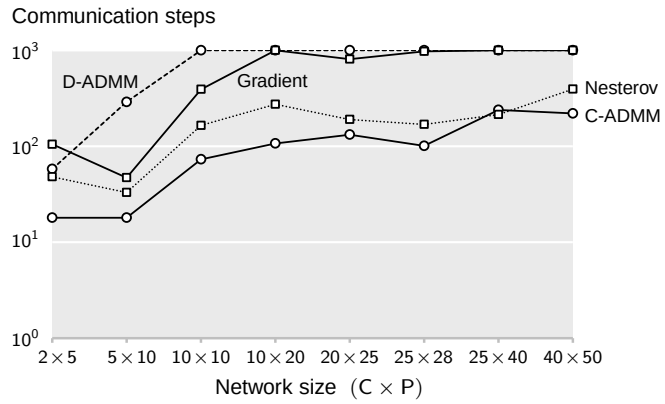
IV. Simulations

We now present some simulation results comparing C-ADMM's performance with other algorithms. We consider the ordinary gradient method, Nesterov's method, and D-ADMM [8], which was designed to solve the more general problem (2), not (1). D-ADMM thus requires exchanging full solution estimates between the nodes. This will be taken in account in our measure of performance: communication steps. A communication step occurs after all nodes exchange their current estimate of the components of $x^\star$. Note that computation time inside each node is about the same in all algorithms and, many times, negligible with respect to the communication times. In D-ADMM, each communication step will count as C communication steps, since the vectors exchanged are C times larger.

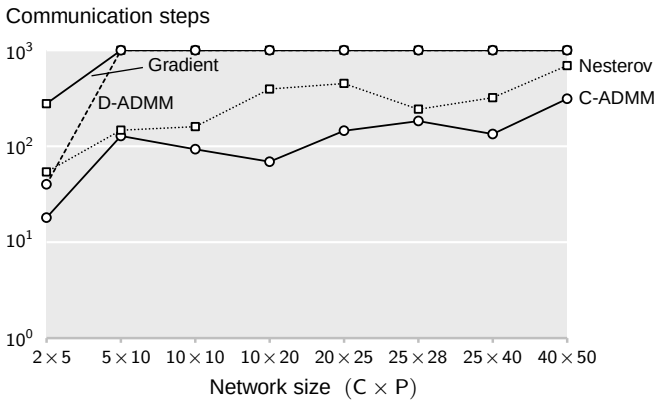
Network models. We generated random bipartite graphs following an Erdős-Rényi model: each pair of nodes (belonging to different groups) is linked with probability p. We considered $p = 0.25$ and $p = 0.75$.

Congestion control. We considered networks with several sizes, ranging from 2×5 to 40×50, where $C\times P$ means that the network has C central nodes and P peripheral nodes. Regarding the weights w_s for each source (central) node, they were generated randomly between 1 and 11. All algorithms stopped whenever $\|x^k - x^\star\|/\|x^\star\| \leq 10^{-3}$, where $x^k = (x_1^k,\ldots,x_C^k)$ is the global estimate at iteration k and $x^\star$ was computed beforehand in a centralized way, or when 10^3 communication steps were achieved. For the augmented Lagrangian algorithms (C-ADMM and D-ADMM), we selected the constant ρ from the set $\{0.01, 0.05, 0.1, 0.5, 1, 5, 10\}$ and considered always the ρ that yielded the best result. A similar scheme was adopted for choosing the Lipschitz constant L for the gradient and Nesterov algorithms, for $L \in \{10^{-2}, 10^{-1}, 1, 5, 10, 15, 20\}$. The reason for "guessing" a Lipschitz constant for these methods is because the gradient of the objective of (19) is not Lipschitz continuous.

Fig. 3 shows the simulation results, depicting the communication steps as a function of the network. These communication steps are counted in the bipartite network (Fig. 2(b)), not in the original network (Fig. 2(a)). Of course, each communication occurring in the bipartite network may correspond to several communications in the original network.



(a) $p = 0.25$



(b) $p = 0.75$

Fig. 3. Results of the simulations for the TCP/IP congestion control application in Erdős-Rényi networks with parameters (a) $p = 0.25$ and (b) $p = 0.75$.

The results for sparsely connected networks, Fig. 3(a), and for densely connected networks, Fig. 3(b), are very similar: the proposed algorithm (C-ADMM) required always less communication steps than competing algorithms, yet it was closely followed by Nesterov's algorithm.

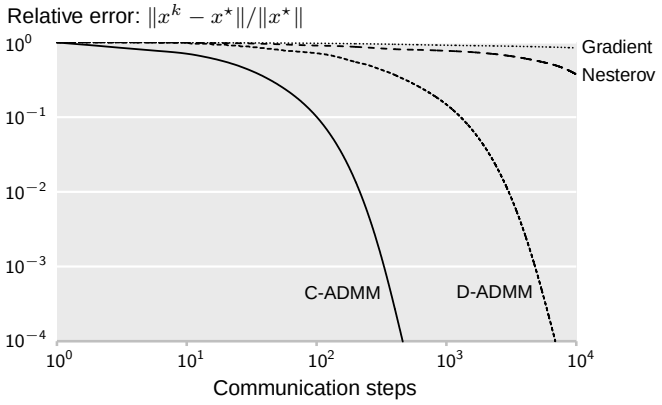


Fig. 4. Results for the MPC in an Erdős-Rényi network with parameter $p = 0.25$. There were $C = 5$ actuators, each with a scalar variable ($m_c = 1$), and $P = 10$ systems, each with a state of dimension $n_p = 2$.

MPC. For the MPC problem, we just present results for the second network of Fig. 3(a), i.e., a 5×10 network created with an Erdős-Rényi model with $p = 0.25$. The results for the other networks of Fig. 3 are very similar. All the vectors and matrices were generated randomly: we set $n_p = 2$, $m_c = 1$, and $T = 10$ for the time horizon. In this case, the Lipschitz constant for the gradient and Nesterov method were computed exactly, and we added 50 and 100 to the set of possible values for ρ.

Fig. 4 shows the results of the simulations by depicting the relative error along the communication steps. C-ADMM required always less iterations than the other algorithms to achieve any value for the relative error. This time both the gradient and Nesterov methods performed poorly; we attribute that to the large Lipschitz constant for this problem, in this case 3×10^7. Only the augmented Lagrangian methods, C-ADMM and D-ADMM, performed well, probably because they are not as sensitive to Lipschitz constant values.

V. Conclusions and Future Work

We proposed an efficient algorithm for solving objective-coupled optimization problems in bipartite networks. The proposed algorithm is based on the Alternating Direction Method of Multipliers and is proved to converge to the same solution as if the problem were solved in a centralized way. We applied the algorithm to distributed model predictive control and to TCP/IP congestion control. Through numerical simulations, we showed that the algorithm is more communication-efficient than previous algorithms, making it attractive to energy-constrained environments.

Future work consists of extending the algorithm to more general networks and to address large-scale problems.

References

[1] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Contr. Sys. Techn.*, vol. 18, no. 2, 2010.

[2] 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.

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

[4] A. Venkat, J. Rawlings, and S. Wright, "Stability and optimality of distributed model predictive control," in *IEEE Conf. Dec. Contr.*, 2005.

[5] M. Chiang, S. Low, A. Calderbank, and J. Doyle, "Layering as optimization decomposition: a mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, 2007.

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

[7] 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. Learning*, vol. 3, no. 1, 2011.

[8] J. Mota, J. Xavier, P. Aguiar, and M. Püschel, "D-ADMM: A communication-efficient distributed algorithm for separable optimization," 2012, submitted, [Online] http://arxiv.org/abs/1202.2805.

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

[10] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Comm.*, vol. 24, no. 8, 2006.

[11] C. Tan, D. Palomar, and M. Chiang, "Distributed optimization of coupled systems with applications to network utility maximization," in *IEEE Inter. Conf. Acoust., Speech, Signal Process.*, 2006.

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

[13] D. Barcelli, D. Bernardini, and A. Bemporad, "Synthesis of networked switching linear decentralized controllers," in *IEEE Conf. Dec. Contr.*, 2010.