Hyper-graph partitioning for a multi-agent reformulation of large-scale MILPs

Lucrezia Manieri *Student Member, IEEE*, Alessandro Falsone *Member, IEEE*, Maria Prandini *Fellow, IEEE*

***Abstract*— This paper addresses the challenge of solving large-scale Mixed Integer Linear Programs (MILPs). A resolution scheme is proposed for the class of MILPs with a hidden constraint-coupled multi-agent structure. In particular, we focus on the problem of disclosing such a structure to then apply a computationally efficient decentralized optimization algorithm recently proposed in the literature. The multi-agent reformulation problem consists in manipulating the matrix defining the linear constraints of the MILP so as to put it in a singly-bordered block-angular form, where the blocks define local constraints and decision variables of the agents, whereas the border defines the coupling constraints. We translate the matrix reformulation problem into a hyper-graph partitioning problem and introduce a novel algorithm which accounts for the specific requirements on the singly-bordered block-angular form to best take advantage of the decentralized optimization approach. Numerical results show the effectiveness of the proposed hyper-graph partitioning algorithm.**

***Index Terms*— Large-scale systems, Optimization, Computational methods.**

I. INTRODUCTION

A Mixed Integer Linear Program (MILP) is an optimization program with both continuous and discrete decision variables of the following form:

$$\min_{x} \quad c^\top x \tag{1a}$$
$$\text{subject to:} \quad Ax \leq b \tag{1b}$$
$$x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} \tag{1c}$$

where vector x collects the n_c continuous decision variables and the n_d discrete ones, vector $c \in \mathbb{R}^{n_c+n_d}$ defines the cost function, matrix $A \in \mathbb{R}^{q \times n_c+n_d}$ and vector $b \in \mathbb{R}^q$ define q scalar constraints. A decision vector x is *feasible* if it belongs to the mixed integer feasibility set $S = \{x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} : Ax \leq b\}$. A decision vector $x \in S$ is an optimal solution to (1) if it minimizes the cost function $c^\top x$ over S.

MILPs arise in different contexts and engineering applications and allow to formulate a variety of decision-making problems involving systems comprising continuous and logical components, [1]–[6]. However, if the system is large-scale and with a high number of discrete decision variables, then, the resulting MILP is typically hard to solve because of its combinatorial complexity: finding an optimal solution is often not viable in practice, and one has to resort to heuristic approaches to recover computational tractability and find a solution that is – at least – feasible, see e.g., [7] and [8]. The same issue has been also addressed in recent works on distributed optimization of constraint-coupled multi-agent MILPs, [9]–[12].

In a constraint-coupled multi-agent MILP, multiple agents cooperatively aim at optimizing the sum of their individual cost functions with respect to local decision variables subject to both individual and global constraints originating from resource sharing. Formally, a constraint-coupled multi-agent MILP is given by:

$$\min_{x_1,\dots,x_m} \quad \sum_{i=1}^{m} c_i^\top x_i \tag{2a}$$
$$\text{subject to:} \quad \sum_{i=1}^{m} E_i x_i \leq f \tag{2b}$$
$$x_i \in X_i,\ i = 1,\dots,m, \tag{2c}$$

where X_i in (2c) is the mixed-integer set defined by the local constraints and given by $X_i = \{x_i \in \mathbb{R}^{n_{c,i}} \times \mathbb{Z}^{n_{d,i}} : D_i x_i \leq d_i\}$, with $n_{c,i}$ and $n_{d,i}$ respectively denoting the number of continuous and discrete decision variables of agent i. The *coupling constraint* in (2b) is defined by vector $f \in \mathbb{R}^p$ and matrices $E_i \in \mathbb{R}^{p \times n_i}$, $i = 1,\dots,m$, where $n_i = n_{c,i} + n_{d,i}$ is the total number of decision variables of agent i.

The work in [11] introduces a decentralized iterative approach for computing a feasible solution to (2) in a finite number of iterations, while quantifying its sub-optimality level. A key point of the decentralized scheme in [11] is that, at each iteration, each agent solves a lower-dimensional MILP, while a central unit handles the coupling constraints. This allows to address multi-agent MILPs with a high number of agents. The strategy is most effective if the number of discrete variables per agent is small, since this determines the computational complexity of the resolution scheme, and if the number of agents is large compared to the number of coupling constraints, since this affects the sub-optimality level of the obtained solution.

The constraint-coupled multi-agent MILP (2) can be rewritten in the general MILP form (1) by defining $c^\top = [c_1^\top \ \cdots \ c_m^\top]$

The authors are with the Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano, Italy, email: name.surname@polimi.it.

and collecting all local and global constraints in

$$\underbrace{\begin{bmatrix} D_1 & & \\ & \ddots & \\ & & D_m \\ E_1 & \cdots & E_m \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} d_1 \\ \vdots \\ d_m \\ f \end{bmatrix}}_{b}, \tag{3}$$

where the resulting matrix A has a *singly-bordered block-angular* structure, [13]. Starting from this observation, we propose in this paper a resolution scheme for a large-scale MILP that uncovers its hidden constraint-coupled multi-agent structure so as to solve it using [11]. In order to find if (1) has such a hidden structure, we need to manipulate the matrix A and reduce it to a singly-bordered block-angular form, where the blocks define both local constraints and local decision variables of the fictitious agents, whereas the border corresponds to the coupling constraints.

Retrieving a block-angular form for a (sparse) matrix A is a well-known problem. For example, in numerical analysis it is used to parallelize computations in LU or QR decomposition, matrix multiplication and inversion, whereas in optimization, it is adopted in the Dantzig-Wolfe decomposition [14]. For this reason, several algorithms have been proposed to perform such a transformation. The most common approach consists in reformulating the matrix transformation problem into a graph partitioning problem. First, a suitable hyper-graph representation for matrix A is derived with decision variables associated to nodes and constraints to connections among nodes, then a partition of the hyper-graph nodes in m sets (parts) with a minimal number of connection between parts (cut-size) is searched for, and, finally, the hyper-graph partition is re-interpreted as a permutation of the matrix A (see e.g. [13] and [15]) leading to a singly-bordered block-angular structure with parts associated with blocks and the cut-size to the border. Unfortunately, available algorithms for hyper-graph partitioning are not directly applicable to our setting, mainly because they do not take into account the following requirements originated from the adoption of the multi-agent resolution scheme in [11]: *i)* the cut-size of the partition (the number of coupling constraints) has to be small compared to the number of parts (the agents) so as to reduce the sub-optimality level; *ii)* nodes associated with discrete decision variables have to be fairly distributed among the parts so as to balance the computational load among the fictitious agents.

In this paper, we shall introduce a novel hyper-graph partitioning algorithm that accounts for these requirements and show its performance through a simulation-based comparative analysis with a state-of-the-art algorithm.

II. Hyper-graph representation of a matrix

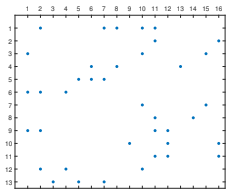
We start recalling some notions on graph theory to then describe the hyper-graph representation of a matrix and the reformulation of the problem of reducing it to a singly-bordered block-angular form as a partitioning problem.

A *hyper-graph* $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ is defined as a set $\mathcal{U}$ of nodes and a set $\mathcal{N}$ of nets (or hyper-edges). Every net $n_i \in \mathcal{N}$ is a subset of nodes, $n_i \subseteq \mathcal{U}$, and represents a connection among them. If all nets have cardinality 2, each (hyper-)edge connects two nodes and the standard notion of graph is recovered. In Figure 1c, we report an example of a hyper-graph $\mathcal{H}$. Each node is represented with a circle containing its label u_j, while each net is depicted as a solid square labelled with n_i. The nodes in a net are called *pins* and the set of pins in a net n_i is denoted as $\mathrm{Pins}(n_i)$. The set of nets connected to a node u_j is instead denoted as $\mathrm{Nets}(u_j)$. A net n_i is said to be *incident* on a node u_j if $u_j \in \mathrm{Pins}(n_i)$, and nodes u_h and u_j are *neighbours* if there exists a net n_i incident on (i.e. connecting) both nodes. The set of neighbours of node u_j is denoted as $\Gamma(u_j)$. For example, net n_3 in Figure 1c is incident on nodes u_1, u_{10} and u_{15}, that are its pins (i.e. $\mathrm{Pins}(n_3) = \{u_1, u_{10}, u_{15}\}$); u_1, u_{10} and u_{15} are neighbours and $\Gamma(u_{15}) = \{u_1, u_{10}\}$; nets n_3, n_6, and n_9 belong to $\mathrm{Nets}(u_1)$. An *m-way* node partition of a hyper-graph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ is a collection $\Pi = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$ defining a partition of the node set $\mathcal{U}$ in m parts. Given a partition Π of $\mathcal{H}$, we say that net n_i *connects* part $\mathcal{U}_h$ if n_i has at least one pin in $\mathcal{U}_h$. Nets are called *cut* (or *external*) if they connect more than one part and *uncut* (or *internal*) otherwise. Note that it is always possible to derive an equivalent representation of Π as an *m-way net partition* $\Pi = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\} \equiv \{\mathcal{N}_1, \ldots, \mathcal{N}_m; \mathcal{N}_{ext}\}$, where $\mathcal{N}_h = \{n \in \mathcal{N} : \mathrm{Pins}(n) \subseteq \mathcal{U}_h\}$ contains the internal nets connecting only part $\mathcal{U}_h$, $h = 1, \ldots, m$, whilst $\mathcal{N}_{ext} = \mathcal{N} \setminus \bigcup_{h=1}^{m} \mathcal{N}_h$ contains all the cut nets. The cardinality of $\mathcal{N}_{ext}$ is the *cut-size* of the partition. Figure 1c shows a 3-way partition of the hyper-graph $\mathcal{H}$, with the ellipses denoting the parts. Its cut-size is $p = 2$, being $\mathcal{N}_{ext} = \{n_1, n_9\}$. A partition $\Pi = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$ is called ε-balanced if each part $\mathcal{U}_h$ satisfies $|\mathcal{U}_h| \leq (1+\varepsilon) \left\lceil \frac{|\mathcal{U}|}{m} \right\rceil$ where $|\mathcal{U}|$ denotes the cardinality of set $\mathcal{U}$. The partition is *balanced* when $\varepsilon = 0$ (see Figure 1c). An ε-balanced m-way partition is *optimal* if it has minimum *cut-size*.

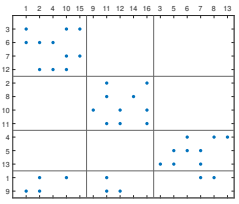
Given a matrix A, we can derive its *row-net hyper-graph* representation by constructing a hyper-graph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, where each node in $\mathcal{U}$ identifies a column of A and each net in $\mathcal{N}$ corresponds to a row of A. More specifically, if the entry a_{ij} of A is non-zero, then node u_j is a pin of net n_i. Since each row of A corresponds to a constraint and each column to a decision variable, then the nodes of the hyper-graph represent the decision variables and each net represents a constraint. Matrix A in Figure 1a, for example, translates into the hyper-graph $\mathcal{H}$ with 16 nodes and 13 nets in Figure 1c. An *m-way* partition $\Pi = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\} \equiv \{\mathcal{N}_1, \ldots, \mathcal{N}_m; \mathcal{N}_{ext}\}$ of $\mathcal{H}$ induces a permutation of A and the permuted matrix A^{Π} has singly-bordered block-angular structure. Specifically, nodes in $\mathcal{U}_h$ and internal nets in $\mathcal{N}_h$ identify columns and rows of block D_h, $h = 1, ..., m$, in (3), whilst nets in $\mathcal{N}_{ext}$ correspond to the border $[E_1, \ldots, E_m]$, [13]. Figure 1b shows the permuted constraint matrix A^{Π} associated to the 3-way partition of $\mathcal{H}$ in Figure 1c.

III. Proposed hyper-graph partitioning method

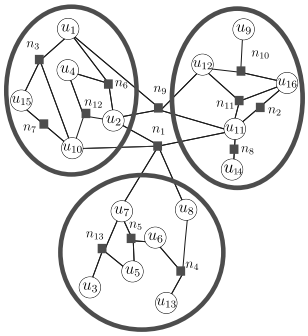
State-of-the-art hyper-graph partitioning algorithms aim at finding an optimal ε-balanced partition, for an a-priori given number m of parts. However, as anticipated at the end of the introduction, for the decentralized multi-agent approach in



(a) Constraint matrix A



(b) Permuted matrix A^{Π}



(c) Hyper-graph $\mathcal{H}$ and a possible 3-way balanced partition $\Pi = \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3\}$

Fig. 1: Constraint matrix with a hidden structure.

[11] to be effective in solving the MILP in (1), we need to bring matrix A into a singly-bordered block-angular through a hyper-graph partitioning strategy that jointly

- minimizes the ratio $\frac{p}{m}$ between the cut-size p and the number of parts m, as opposed to p only;
- balances the distribution among the parts of the nodes corresponding to the discrete decision variables (*discrete nodes*).

As for nodes corresponding to continuous decision variables (*continuous nodes*), they can be unevenly distributed if this helps in balancing the discrete nodes and/or reducing $\frac{p}{m}$.

In this paper, we propose a method that integrates a suitably designed iterative algorithm for partitioning a hyper-graph in m parts (Section III-A) within a strategy to determine m so as to minimize the ratio $\frac{p}{m}$ (Section III-B).

A. Hyper-graph partitioning in m parts

Algorithm 1 provides the pseudo-code of the proposed procedure for partitioning in a given number m of parts a hyper-graph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ representing matrix A appearing in a MILP (1). Besides m and the sets $\mathcal{U}_c$ and $\mathcal{U}_d$ of continuous and discrete nodes in $\mathcal{U}$, Algorithm 1 takes as inputs the maximum number $D_{\max}$ of discrete nodes per part and the maximal value η for the ratio $\frac{p}{m}$ where p is the cut-size of the sought hyper-graph partition. Since our final objective is solving (1) through the multi-agent scheme [11], then, $D_{\max}$ is dictated by the MILP solver capability, whereas $\eta \in (0, 1)$ relates to the admissible degradation of the quality of the solution. Note that m must satisfy $mD_{\max} \geq |\mathcal{U}_d|$, since otherwise some of the parts in the partition would necessarily contain a number of discrete nodes that exceeds the solver capability.

An m-way partition $\Pi^\star$ of $\mathcal{H}$ is determined in Algorithm 1 by subsequently isolating from $\mathcal{H}$ suitably chosen, weakly coupled, groups of nodes (the parts) and adding them to $\Pi^\star$. More precisely, a hyper-graph $\mathcal{H}_\rho$ is first introduced (cf. Step 4) by adding to the original hyper-graph $\mathcal{H}$ a number $n_\delta = mD_{\max} - n_d$ of discrete dummy nodes, i.e., nodes not connected by any net. As suggested in [16], this allows to

Algorithm 1: Hyper-graph partitioning

Input: $\mathcal{H}$ original hyper-graph; $\mathcal{U}_d$ set of discrete nodes; $\mathcal{U}_c$ set of continuous nodes; $D_{\max}$ max discrete nodes; m number of parts; η threshold for the $\frac{p}{m}$ ratio

1: $h = m$
2: $\mathcal{H}_\rho = (\mathcal{U}_\rho, \mathcal{N}_\rho) = \mathcal{H} = (\mathcal{U}_c \cup \mathcal{U}_d, \mathcal{N})$
3: $n_\delta = m \cdot D_{\max} - |\mathcal{U}_d|$
4: $\mathcal{H}_\rho$ = `add_dummy_nodes`$(\mathcal{H}_\rho, n_\delta)$
5: $\Pi^\star = \emptyset$
6: $p = 0$
7: **repeat**
8: $\quad\widetilde{\mathcal{H}}_\rho$ = `coarsening`$(\mathcal{H}_\rho)$
9: $\quad\widetilde{\Pi}_\rho = \{\mathcal{U}_{\rho,i}\}_{i=1}^{h} = \{\{\mathcal{N}_{\rho,i}\}_{i=1}^{h}; \mathcal{N}_{\rho,ext}\} =$ `h-way_partitioning`$(\widetilde{\mathcal{H}}_\rho, h)$
10: $\quad[\mathcal{H}_\rho, \Pi_\rho]$ = `ungrouping`$(\widetilde{\mathcal{H}}_\rho, \widetilde{\Pi}_\rho)$
11: $\quad[\mathcal{U}_{\rho,\beta_1}, \ldots, \mathcal{U}_{\rho,\beta_B}, \mathcal{N}_{\rho,\tau}]$ = `parts_isolation`$(\mathcal{H}_\rho, \Pi_\rho)$
12: $\quad\mathcal{H}_\rho = \left(\mathcal{U}_\rho \setminus \left\{\bigcup_{b=1}^{B} \mathcal{U}_{\rho,\beta_b}\right\}, \mathcal{N}_\rho \setminus \left\{\bigcup_{k=1}^{B} \mathcal{N}_{\rho,\beta_k}\right\} \setminus \mathcal{N}_{\rho,\tau}\right)$
13: $\quad h \leftarrow h - B$
14: $\quad\Pi^\star \leftarrow \Pi^\star \cup \{\mathcal{U}_{\rho,\beta_1}, \ldots, \mathcal{U}_{\rho,\beta_B}\}$
15: $\quad p \leftarrow p + |\mathcal{N}_{\rho,\tau}|$
16: **until** $h < 2 \vee \frac{p}{m} > \eta$
17: $\Pi^\star$ = `remove_dummy_nodes`$(\Pi^\star)$

Output: Partition $\Pi^\star$

add some flexibility while seeking for an m-way partition of $\mathcal{H}_\rho$ that is balanced in terms of discrete nodes, since those parts with dummy nodes will have an actual number of discrete nodes smaller than the other parts but in any case not exceeding the allowed $D_{\max}$ value. The underlying idea is that we are certainly looking for a balanced partition, but we allow for imbalance in those cases when this is needed to meet the constraint of not exceeding the solver capabilities $D_{\max}$ while having a low $\frac{p}{m}$ ration.

Each iteration is performed on $\mathcal{H}_\rho$ whose size gets progressively reduced according to the following steps. The initial *coarsening* (cf. Step 8) phase assigns each continuous node in the hyper-graph $\mathcal{H}_\rho$ to the discrete node it is most coupled with, so as to enforce in the final partition a balance in the discrete nodes distribution. The resulting coarsened hyper-graph $\widetilde{\mathcal{H}}_\rho$ contains *supernodes*, each one with a discrete node only and the associated continuous ones. Then, a *partitioning* operation (cf. Step 9) computes an h-way balanced minimum cut-size partition $\widetilde{\Pi}_\rho$ of the coarsened hyper-graph $\widetilde{\mathcal{H}}_\rho$. The number h of desired parts is set equal to m at the first iteration and then progressively reduced as groups of nodes are removed (cf. Step 13). Last, the *isolation* step (cf. Step 11) is performed to start isolating a block of the final singly-bordered block-angular form of A and remove it while accounting for its contribution to the size p of the border. To this purpose, supernodes are ungrouped (cf. Step 10) to translate $\widetilde{\Pi}_\rho$ in terms of the corresponding partition Π_ρ of the original hyper-graph $\mathcal{H}_\rho$, which may be different from $\mathcal{H}_\rho$ since it accounts also for indirect connections between discrete nodes through continuous nodes. Those parts of Π_ρ that are connected with the rest of the hyper-graph by the smallest number of external nets are then isolated at Step 11, and the set $\mathcal{N}_{\rho,\tau}$ containing the external nets that connect them to the rest of the graph is defined. Note that the number B of parts that are isolated at Step 11 can be larger than 1. This is because, while identifying

a least-connected part and removing the nets that connect it to the others, further ones that are not connected to the rest of the hyper-graph may arise, so that also those parts can be removed without increasing the size p of the border. A final *removal* (cf. Step 12) step removes from the hyper-graph $\mathcal{H}_\rho$ the B least-connected parts that have been isolated, together with the external nets connecting them to the rest of the hyper-graph in set $\mathcal{N}_{\rho,\tau}$ and their internal nets in $\mathcal{N}_{\rho,\beta_k}$, $k = 1,\dots,B$.

At each iteration, the identified B least-connected parts are stored in the partition $\Pi^\star$ (cf. Step 14) whose cut-size p is updated accordingly (cf. Step 15), thus growing throughout iterations. Algorithm 1 is either run until m parts are identified or preemptively interrupted if the ratio $\frac{p}{m}$ exceeds the threshold η. As better clarified next, the coarsening and partitioning steps are not guaranteed to be optimal and their outcome can vary in different runs. This motivates the adoption of subsequent coarsening and partitioning steps within Algorithm 1 not to get stuck in a sub-optimal solution, as well as its repetitive application to improve the current solution or to find one in case of an early stop.

A more detailed description of the procedures implementing the first three steps is provided hereafter.

1) Coarsening: The grouping strategy proposed to associate continuous variables to discrete ones is summarized in Algorithm 2, and is a modified version of the *Edge Coarsening (EC)* procedure adopted in [17]–[19]. EC merges pairs of nodes based on the strength of their connection, evaluated considering the number and the size of the nets they *share* (i.e. the nets that are incident on both nodes). First, for each net n in the hyper-graph a *hyper-edge weight* $\omega(n) = \frac{1}{|\mathrm{Pins}(n)|}$ is computed. Then, the strength of the connection between a pair of nodes u, v is measured by the *rating function*

$$r_{\mathrm{EC}}(u,v) = \sum_{n \in \mathrm{Nets}(u) \cap \mathrm{Nets}(v)} \omega(n)$$

which can be used to determine, for each node u, the best *contraction partner* $v^\star \in \arg\max_v r_{\mathrm{EC}}(u,v)$ to be merged with, the rationale being that the smaller the size of the net n and the more nets connecting u and v, the tighter is the coupling between its pins. The grouping operation is, then, performed by randomly selecting a node u in the hyper-graph, searching for its best contraction partner $v^\star$ among its neighbours $\Gamma(u)$ and merging them together in a supernode. The procedure is repeated until a given criterion on the coarsened hyper-graph is met.

The grouping strategy proposed in Algorithm 2 follows a similar paradigm, with three modifications: i) it never merges discrete nodes together; ii) it merges continuous nodes preferably with discrete nodes, possibly with other continuous nodes but only if they are neighbours; iii) it uses a modified rating function $r(u,v) = \frac{r_{\mathrm{EC}}(u,v)}{\nu(u)+\nu(v)}$, where the weight $\nu(u)$ is equal to the number of nodes inside the (super)node u (1 if u is standard node) and the scaling factor $\frac{1}{\nu(u)+\nu(v)}$ is introduced to promote a balanced distribution of the continuous variables in the supernodes. Deterministic tie-break rules are applied in case of multiple maximizers.

2) h-way partitioning: The h-way partitioning applied to the coarsened hyper-graph $\widetilde{\mathcal{H}}_\rho$ distributes the supernodes (each one associated with a discrete, possibly dummy, node) among the parts following a *sequential break-off* paradigm. The procedure is summarized in Algorithm 3.

Algorithm 2: Hyper-graph coarsening

Input: $\mathcal{H}$ hyper-graph

1: $\widetilde{\mathcal{H}} = \mathcal{H} = (\mathcal{U}_c \cup \mathcal{U}_d, \mathcal{N})$
2: **for each** $v \in \mathcal{U}_c$ **do**
3: **if** $\Gamma(v) \cap \mathcal{U}_d \neq \emptyset$ **then**
{find the best discrete contraction partner}
4: select $u^\star \in \arg\max_{u \in \Gamma(v) \cap \mathcal{U}_d} r(u,v)$
5: **else if** $\Gamma(v) \cap \mathcal{U}_c \neq \emptyset$ **then**
{find the best continuous contraction partner}
6: select $u^\star \in \arg\max_{u \in \Gamma(v) \cap \mathcal{U}_c} r(u,v)$
7: **else**
{assign to a random discrete node}
8: randomly select $u^\star \in \mathcal{U}_d$
9: **end if**
10: $\widetilde{\mathcal{H}} = \mathtt{contract}(\widetilde{\mathcal{H}}, u^\star, v)$
11: **end for**

Output: $\widetilde{\mathcal{H}}$ coarsened hyper-graph

First introduced in [16], sequential break-off finds an h-way partition of an input hyper-graph with n nodes by iteratively isolating groups of $\frac{n}{h}$ nodes having a weak connection (i.e., a low number of common nets) with the remaining ones. More formally, at the first iteration, an initial 2-way partition (bisection) $\{\mathcal{U}_1, \mathcal{U} \setminus \mathcal{U}_1\}$ is created (cf. Steps 4-5) by randomly selecting a subset $\mathcal{U}_1$ of $n_h = \lceil \frac{n}{h} \rceil$ nodes. Such bisection is, then, optimized (cf. Step 6) to reduce the cut-size preserving the relative imbalance, the refined subset $\mathcal{U}_1^\star$ is set aside and the procedure is repeated on the remaining elements in $\mathcal{U} \setminus \mathcal{U}_1^\star$ until all h parts of the h-way partition are stored in Π_0. Finally, the procedure performs pair-wise comparisons between the parts (cf. Step 11) aiming at making the h-way partition *pair-wise optimal*, i.e. such that for every pair $\mathcal{U}_i, \mathcal{U}_j$, $i \neq j$, the 2-way partition $\{\mathcal{U}_i, \mathcal{U}_j\}$ is a balanced minimum cut-size bisection of the sub-hyper-graph induced by $\mathcal{U}_i$ and $\mathcal{U}_j$. Comparisons are performed until no improvement can be made or a maximum number of comparisons is reached. The minimum cut-size bisection returned by Step 6 is obtained starting from the initial random bisection and then applying the state-of-the-art *iterative refinement* scheme proposed by Fiduccia and Mattheyses (FM) in [20], which is commonly adopted, with minor modifications, in most of the available partitioning algorithms.

3) Isolation: The part isolation procedure works as follows. We first determine the number of cut nets connecting each part $\mathcal{U}_{\rho,i}$, $i = 1,\dots,h$. We then select the part $\mathcal{U}_{\rho,\beta_1}$ having the minimum number of nets in the cut. $\mathcal{U}_{\rho,\beta_1}$ is then added to the output partition $\Pi^\star$ and removed from $\mathcal{H}_\rho$ together with its internal and external nets. All other blocks $\mathcal{U}_{\beta_2},\dots,\mathcal{U}_{\beta_B}$ (if any) that after this removal operation are no longer connected by any external net are also added to the final partition $\Pi^\star$. This allows the next iteration to seek for $(h - B)$ instead of $h - 1$ parts of the reduced hyper-graph $\mathcal{H}_\rho$, speeding up the decomposition without affecting the cut-size of the final partition $\Pi^\star$.

Algorithm 3: h-way partitioning (via sequential break-off)

Input: $\mathcal{H}$ hyper-graph, h number of parts

$n_h = \left\lceil \frac{|\mathcal{U}|}{h} \right\rceil$
$\mathcal{U}_2 = \mathcal{U}$
for $k = 1, \ldots, h-1$ **do**
  $\mathcal{U}_1 = \texttt{select\_n\_h\_elements}(\mathcal{U}_2, n_h)$
  $\mathcal{U}_2 = \mathcal{U}_2 \setminus \mathcal{U}_1$
  $\{\mathcal{U}_1^\star, \mathcal{U}_2^\star\} = \texttt{FM\_refinement}(\mathcal{H}, \{\mathcal{U}_1, \mathcal{U}_2\})$
  $\mathcal{U}_{\Pi_0,k} = \mathcal{U}_1^\star$
  $\mathcal{U}_2 = \mathcal{U}_2^\star$
end for
$\Pi_0 = \{\mathcal{U}_{\Pi_0,1}, \ldots, \mathcal{U}_{\Pi_0,h-1}, \mathcal{U}_2\}$
$\Pi^\star = \texttt{pairwise\_comparison}(\Pi_0, \mathcal{H})$

Output: $\Pi^\star$ final partition

B. Estimation of the number m of parts

Algorithm 1 partitions the input hyper-graph $\mathcal{H}$ in a number of parts m that is fixed and a-priori given. The overall decomposition procedure, however, should be able to provide an estimate for m, maximizing the number of parts while preserving a satisfactory cut-size p. We propose here a strategy for estimating m that avoids running Algorithm 1 for all values for m in a range, which can be ineffective and time-consuming. This is achieved by exploiting the flexibility introduced by adding discrete dummy nodes. Recall that we introduced in Algorithm 1 n_δ dummy nodes to allow the retrieval of partitions with unbalanced distributions of the actual discrete nodes among the parts. When the number of dummy nodes is sufficiently large, the procedure can use some of them to entirely fill one or more *surplus* parts. Whenever this occurs, the initial estimate $\hat{m}$ can be reduced, and Algorithm 1 can be re-launched with the new guess.

The procedure starts by seeking a m-way partition with $m = \hat{m}_0 = \left\lceil \frac{n_d}{D_{\min}} \right\rceil$, where $D_{\min} \in (1, D_{\max}]$ represents an affordable computational effort that can be easily endured by each agent, in terms of number of discrete variables. At each iteration k, the new $\hat{m}_{k+1}$ is computed deducting from $\hat{m}_k$ the number of surplus parts identified in the partitioning phase, till possibly $\left\lceil \frac{n_d}{D_{\max}} \right\rceil$ is reached. The returned partition is the one that minimizes the $\frac{\hat{p}}{\hat{m}}$ ratio. Notice that the efficiency of such estimation strategy strongly depends on the ability of the partitioning algorithm to create empty parts that other state-of-the-art partitioning schemes do not provide.

IV. Comparative simulation analysis

In this section we assess the performance of the proposed hyper-graph partitioning method on a set of 100 matrices A with hidden singly-bordered block angular structure A^Π. The matrices are generated according to the following protocol: the number of columns of A (and A^Π) associated with continuous and discrete nodes of its hyper-graph representation, n_c and n_d, are randomly selected in the intervals $[450, 520]$ and $[400, 500]$, respectively. The number of blocks m° of the hidden matrix A^Π varies between 13 and 20 with uniform probability. The size of the border p° is chosen so as $\frac{1}{0.25}(\frac{p^\circ}{m^\circ} - 0.1)$ has a folded standard normal distribution. Matrix A^Π has a sparsity pattern determined by the density of each block and of the border, ranging respectively in $[0.07, 0.1]$ and $[0.03, 0.2]$. Depending on the distribution of continuous and discrete nodes among the blocks, the hidden structure can be either: a) *perfectly balanced* when both continuous and discrete nodes are equally distributed, b) *only-discrete balanced* when only the discrete nodes are equally distributed, and c) *unbalanced* when discrete nodes are unevenly distributed. The three configurations can occur with the same probability. The amount of imbalance is determined by setting the difference between the maximum and minimum number of discrete and continuous nodes per block equal to $\Delta_d = \lceil \delta_d \cdot \frac{n_d}{m} \rceil$ and $\Delta_c = \lceil \delta_c \cdot \frac{n_c}{m} \rceil$ where $\delta_c \in [0, 0.7]$ and $\delta_d \in [0, 0.5]$ are selected at random. The overall number of rows of A^Π can vary between 60% and 110% of the number of columns. The number of rows per block is chosen to be the same in case a) of perfectly balanced distribution, whilst a maximum difference of 15 is admitted for cases b) and c). A row with coefficients that are nonzero within a block and zero outside is added per block so as to guarantee that it is not possible to move variables from one block to another without increasing the size of the border while preserving the number of blocks.

For comparative purposes, we apply h-Metis, a state-of-the-art hyper-graph partitioning algorithm introduced in [17], that uses subsequent bisections steps to seek for the minimum cut-size partition. We first evaluate the performance of both partitioning approaches in the case when some information on the hidden structure is available and can be used to tune the algorithms. In particular, in our algorithm we use the minimum and maximum number of discrete nodes per part to set the value $D_{\min}$, $D_{\max}$, whilst we let it estimate the number of parts $\hat{m}$. As for h-Metis, we set $m = m^\circ$ and tune the so-called *ub-factor* μ, that fixes the maximum relative imbalance allowed between the parts at each level of the recursive bisection, so as to match the actual imbalance. Indeed, according to [21], each part of the final partition can contain a number of nodes ranging from $\left(\frac{50-\mu_b}{100}\right)^l \cdot |\mathcal{U}|$ to $\left(\frac{50+\mu_b}{100}\right)^l \cdot |\mathcal{U}|$, where $l = \lceil \log_2(m^\circ) \rceil$ is the number of bisection steps. Thus we set μ as the minimum value for which the lower-bound is smaller than $\min\{n_i\}$ and the upper-bound is greater than $\max\{n_i\}$, n_i being the number of nodes of part i of the hidden partition. We also combine h-Metis with the proposed strategy for the estimation of m. The algorithm is applied again on the matrices and it is granted the same flexibility given to our procedure by adding the same amount n_δ of dummy nodes.

Both algorithms are run twice for each matrix instance, and the outcome of the run with the lowest $\frac{\hat{p}}{\hat{m}}$ estimate is considered. The hyper-graph partitions obtained by the proposed procedure and h-Metis are compared based on their $\frac{\hat{p}}{\hat{m}}$ estimates, denoted as R_p and R_h, respectively. Table I provides a comparison of the results where instances are divided according to their structure in terms of distribution of the continuous and discrete nodes. As shown in the last row of Table I, our algorithm has a better performance ($R_p < R_h$) when h-Metis uses the correct m (left side of the table) and also when it tries to estimate it (right side of the table). If

underlying structure	h-Metis with $m = m^\circ$			fraction of matrices per structure	h-Metis with m estimation		
	$R_p < R_h$	$R_p > R_h$	$R_p = R_h$		$R_p < R_h$	$R_p > R_h$	$R_p = R_h$
balanced	6%	0%	17%	23%	6%	0%	17%
only-discrete balanced	27%	1%	17%	45%	30%	11%	14%
unbalanced	18%	8%	6%	32%	29%	3%	0%
all cases	51%	9%	40%	100%	65%	4%	31%

TABLE I: Comparative analysis in terms of ratio between cut-size and number of parts of our algorithm (R_p) and h-Metis (R_h) using the correct number of parts (left side) or estimating it (right side), on a pool of 100 matrices with different structure as specified in the first column. Each cell reports the percentage with respect to the total number of matrices.

we focus on the first setting, the two algorithms show similar performance in the perfectly balanced and unbalanced cases, whereas our algorithm has a better performance in the only-discrete balanced case, where it can take full advantage of its coarsening procedure. However, as we move to the second setting, our algorithm clearly outperforms h-Metis in both the only-discrete balanced and the unbalanced cases. In order to assess the performance of our algorithm in terms of capability of partitioning evenly the discrete variables among blocks, we focused on the 68 matrix instances with the perfectly balanced and only-discrete balanced structures, set $D_{\min} = 23$ and $D_{\max} = 41$ (so as to allow for some imbalance in the recovered singly-bordered block-angular structure for A), and computed the resulting amount of imbalance ε in terms of discrete variables. The average imbalance over all 68 matrices turned out to be 0.439 against 0.737 obtained using h-Metis with the estimation of the number of blocks. Only 15 out of the 68 singly-bordered block-angular matrices recovered by h-Metis comply with the solver limits, while the other 53 violate the prescribed $D_{\max}$ upper bound on the number of discrete variables per block.

V. CONCLUSION

This work addresses the problem of finding a constraint-coupled multi-agent reformulation for a large-scale MILP with a sparse constraint matrix, so as to make it suited for the resolution via distributed optimization algorithms. Key to the effectiveness of the approach is a fair, possibly even distribution of discrete decision variables among the fictitious agents, and the minimization of the number of coupling constraints. After translating the reformulation problem into a hyper-graph partitioning one, we introduced a novel algorithm to account for the specific requirements of our settings. A comparative analysis of the introduced algorithm versus a state-of-the-art partitioning method was performed via extensive simulations on multiple matrices with a hidden singly-bordered block-angular structure generated at random.

A challenging topic is how to handle the case when the MILP does not have really a hidden multi-agent structure but its constraint matrix can be reduced to a form which is close to a singly-bordered block-angular one. What if the weak connections between agents are neglected when computing the MILP solution? Answering this question requires additional effort, which goes far beyond this work.

REFERENCES

[1] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics and constraints," *Automatica*, vol. 35, pp. 407–427, 1999.

[2] M. Hejri and A. Giua, "Hybrid modeling and control of switching DC-DC converters via MLD systems," in *IEEE Int. Conf. on Automation Science and Engineering*, 2011, pp. 714–719.

[3] A. Parisio, E. Rikos, and L. Glielmo, "A model predictive control approach to microgrid operation optimization," *IEEE Trans. on Control Systems Technology*, vol. 22, no. 5, pp. 1813–1827, 2014.

[4] A. Bemporad, D. Mignone, and M. Morari, "Moving horizon estimation for hybrid systems and fault detection," in *American Control Conf.*, vol. 4, 1999, pp. 2471–2475.

[5] A. Bemporad, F. Borrelli, and M. Morari, "Piecewise linear optimal controllers for hybrid systems," in *American Control Conf.*, vol. 2, 2000.

[6] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in *47th IEEE Conf. on Decision and Control*, 2008.

[7] D. Bertsekas, G. Lauer, N. Sandell, and T. Posbergh, "Optimal short-term scheduling of large-scale power systems," *IEEE Trans. on Automatic Control*, vol. 28, no. 1, pp. 1–11, 1983.

[8] N. J. Redondo and A. Conejo, "Short-term hydro-thermal coordination by lagrangian relaxation: solution of the dual problem," *IEEE Trans. on Power Systems*, vol. 14, no. 1, pp. 89–95, 1999.

[9] R. Vujanic, P. M. Esfahani, P. Goulart, S. Mariéthoz, and M. Morari, "A decomposition method for large scale MILPs, with performance guarantees and a power system application," *Automatica*, vol. 67, 2016.

[10] A. Falsone, K. Margellos, and M. Prandini, "A distributed iterative algorithm for multi-agent milps: Finite-time feasibility and performance characterization," *IEEE Control Systems Letters*, vol. 2, no. 4, pp. 563–568, 2018.

[11] A. Falsone, K. Margellos, and M. Prandini, "A decentralized approach to multi-agent MILPs: finite-time feasibility and performance guarantees," *Automatica*, vol. 103, pp. 141–150, 2019.

[12] A. Testa, A. Rucco, and G. Notarstefano, "Distributed mixed-integer linear programming via cut generation and constraint exchange," *IEEE Trans. on Automatic Control*, vol. 65, no. 4, pp. 1456–1467, 2020.

[13] C. Aykanat, A. Pinar, and U. Catalyurek, "Permuting sparse rectangular matrices into block-diagonal form," *SIAM Journal on Scientific Computing*, vol. 25, 12 2002.

[14] M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. E. Lübbecke, E. Malaguti, and E. Traversi, "Automatic Dantzig–Wolfe reformulation of mixed integer programs," *Mathematical Programming*, vol. 149, no. 1-2, pp. 391–424, 2015.

[15] U. Catalyurek, "A fine-grain hypergraph model for 2d decomposition of sparse matrices," in *15th Int. Parallel and Distributed Processing Symposium*, 2001, pp. 1199–1204.

[16] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[17] G. Karpys, R. Aggarwal, V. Kumar, and S. Shekar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 1, pp. 69–78, 03 1999.

[18] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 8, pp. 655–667, 1998.

[19] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz, "K-way hypergraph partitioning via n-level recursive bisection," in *18th Workshop on Algorithm Engineering and Experiments*, 2016, pp. 53–67.

[20] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *19th Design Automation Conference*, 1982, pp. 175–181.

[21] G. Karypis and V. Kumar, "A hypergraph partitioning package," *Army HPC Research Center, Department of Computer Science & Engineering, University of Minnesota*, 1998.