POLITECNICO
MILANO 1863

School of Industrial and Information Engineering

Master of Science in Automation and Control Engineering

Master Thesis

Large-scale MILP solution via a multi-agent reformulation

Supervisor: **Prof. Maria Prandini**
Co-Supervisor: **Prof. Alessandro Falsone**

Author: **Lucrezia Manieri**

ID:**920062**

A.Y. 2019–2020

To my aunt Mariella,
I've had the time of my life fighting dragons with you.

Acknowledgments

Many people have taught, encouraged, supported, helped and advised me during the time in which I worked on this thesis. I wish I could express my deepest gratitude to each and one of them.

My first thanks go to my supervisor, Professor Maria Prandini, and my co-supervisor, Alessandro Falsone, for guiding and supporting me with (a lot of) patience during the writing of this thesis and beyond. Without them this work would never have seen the light as it is today.

I would also like to thank my travel companion Andrea for him has made these years spent together at the Politecnico brighter. I will always be in his debt, for the help, support and the valuable advice he gave me while writing this thesis and during all the (far too many) projects we worked on together.

I am grateful to my hometown friends, who have been with me for more than ten years now. To Chiara, blunt but capable of great love who has always encouraged me and that believes in me even more than she should. To Giulia, lost and found *"contenta dei deserti"* that showed me how to bend without breaking. To Sara, who has now resigned herself to the fact that her (kilometer-long) voice messages cannot but remain unheard for at least a week. To Caterina, my faithful sister in the daily fight against the patriarchy. And to Veronica, who loves me despite (and perhaps because of) the long chats on summer evenings about the meaning of life.

Finally, I must express my deep gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my

years of study and for being so patient and understanding through the process of researching and writing process of this thesis. None of this would have been possible without them. Thank you.

Ringraziamenti

Molte persone mi hanno insegnato, incoraggiato, sostenuto, aiutato e consigliato durante il tempo in cui ho lavorato a questa tesi. Vorrei poter esprimere la mia più profonda gratitudine a ciascuno di loro.

Il mio primo grazie va alla mia relatrice, la Professoressa Maria Prandini, e al mio co-relatore, Alessandro Falsone, per avermi guidato e sostenuto con (molta) pazienza durante la stesura di questa tesi e non solo. Senza di loro questo lavoro non avrebbe mai visto la luce per come è oggi.

Vorrei, poi, ringraziare il mio compagno di viaggio Andrea per aver illuminato questi anni passati insieme al Politecnico. Sarò per sempre in debito con lui, per l'aiuto, il supporto (emotivo e non) e i preziosi consigli che mi ha dato durante la scrittura di questa tesi e negli svariati (decisamente troppi) progetti a cui abbiamo lavorato insieme.

Sono grata alle mie amiche *"di giù"*, che mi sono accanto da più di dieci anni. A Chiara, schietta ma capace di un grande amore che da sempre mi incoraggia e crede in me molto più di quanto dovrebbe. A Giulia, perduta e ritrovata *contenta dei deserti* che mi ha mostrato come piegarmi di fronte alle difficoltà della vita, senza spezzarmi. A Sara, che si è ormai rassegnata al fatto che i suoi (chilometrici) messaggi vocali rimangano inascoltati per almeno una settimana. A Caterina, mia fedele sorella nella lotta quotidiana contro il patriarcato. E a Veronica, che mi vuole bene nonostante (e forse soprattutto per) le lunghissime chiacchierate nelle sere d'estate sul senso della vita.

Infine, devo esprimere la mia profonda gratitudine ai miei genitori

per avermi fornito un sostegno incessante e un incoraggiamento continuo nel corso della mia carriera universitaria e per essere stati così pazienti e comprensivi durante il processo di ricerca e scrittura di questa tesi. Nulla di questo sarebbe stato possibile senza di loro. Grazie.

Lucrezia

Milan, October 2020

The Devil's in the details.

Contents

Abstract

Mixed Integer Linear Programs (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. If the system is large-scale, however, the resulting MILP is typically hard to solve because of its combinatorial complexity due to the presence of integer decision variables: 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.

This issue has been also addressed in recent works on decentralized optimization for multi-agent systems. In particular, an approach has been proposed for computing a feasible solution to constraint-coupled multi-agent MILPs, while quantifying its sub-optimality level.

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. The introduced decentralized strategy reduces the overall large-scale MILP to multiple lower-dimensional ones (one per agent) that are repeatedly solved a finite number of times by the agents while exchanging information on the coupling constraints. The strategy is most effective if the number of agents is large compared to the number of coupling constraints.

In this thesis, we propose a resolution scheme for a large-scale MILP

with a hidden constraint-coupled multi-agent structure: such a structure is recovered first, and then the previously mentioned decentralized optimization scheme is applied. In order to disclose the hidden constraint-coupled multi-agent structure, we need to manipulate the matrix defining the linear constraints 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.

We translate the matrix reformulation problem into a hyper-graph partitioning problem and introduce a novel partitioning algorithm that accounts for the specific requirements of our setting: i) maximize the number of fictitious agents while minimizing the border size, and ii) evenly distribute the discrete decision variables among the agents so as to uniformly split the computational load.

Performance of the novel partitioning strategy in terms of quality of the obtained result, sensitivity with respect to user-defined parameters, and computational time is assessed through extensive simulations.

Keywords: Large-Scale MILP, Optimization, Decomposition, Hyper-graph partition, Singly-Bordered Matrix

Sommario

I programmi lineari misto-interi (MILPs) si presentano in diversi contesti e applicazioni ingegneristiche e permettono di formulare una grande varietà di problemi decisionali che coinvolgono sistemi formati da componenti continue e discrete. Tuttavia, nel caso di sistemi di larga scala il MILP risultante è tipicamente difficile da risolvere a causa della sua complessità combinatoria, dovuta alla presenza di variabili decisionali intere: spesso trovare una soluzione ottima non è possibile in pratica ed è necessario ricorrere ad approcci euristici per recuperare la trattabilità computazionale e trovare una soluzione che sia almeno ammissibile, cioè compatibile con tutti i vincoli.

Questo problema è stato affrontato anche in articoli recenti relativi all'ottimizzazione decentralizzata per sistemi multi-agente. In particolare, è stato proposto un approccio per calcolare una soluzione ammissibile per MILP multi-agente con vincoli di accoppiamento, quantificando il suo livello di sub-ottimalità.

In un MILP multi-agente con vincoli di accoppiamento, gli agenti collaborano per ottimizzare la somma delle loro funzioni di costo individuali, agendo sulle rispettive variabili decisionali locali e rispettando sia vincoli individuali che globali, originati dalla condivisione delle risorse. La strategia decentralizzata introdotta riduce il MILP globale di larga scala a un insieme di sotto-problemi di dimensione inferiore (uno per agente) che vengono risolti ripetutamente un numero finito di volte dagli agenti, scambiando informazioni sui vincoli di accoppiamento. Tanto maggiore è il numero di

agenti rispetto al numero di vincoli di accoppiamento, tanto più efficace la strategia.

In questa tesi, proponiamo uno schema di risoluzione per MILP di larga scala che hanno una struttura nascosta multi-agente con vincoli di accoppiamento: tale struttura viene prima evidenziata, per poi applicare lo schema di ottimizzazione decentralizzato citato in precedenza. Al fine di rivelare la struttura multi-agente nascosta, è necessario manipolare la matrice che definisce i vincoli lineari e ridurla a una forma blocco-angolare a bordo singolo, dove i blocchi definiscono sia i vincoli locali che variabili di decisione locali degli agenti fittizi, mentre il bordo corrisponde ai vincoli di accoppiamento.

Il problema di riformulazione della matrice viene tradotto in un problema di partizionamento di un iper-grafo e viene introdotto un nuovo algoritmo di partizionamento che tiene conto dei requisiti specifici dello scenario considerato: i) massimizzare il numero di agenti fittizi riducendo al minimo la dimensione del bordo e ii) distribuire uniformemente le variabili decisionali discrete tra gli agenti in modo da dividere uniformemente il carico computazionale.

Le prestazioni della nuova strategia di partizionamento vengono valutate in simulazione, in termini di qualità del risultato ottenuto, sensibilità rispetto ai parametri definiti dall'utente e tempo di calcolo.

Parole chiave: MILP su larga scala, Ottimizzazione, Decomposizione, Partizione di Iper-grafi, Matrici blocco-angolari a bordo singolo

Chapter 1

Introduction

1.1 Context and goal

Technological advances have enabled the introduction of engineering systems with enhanced capabilities in many fields, such as manufacturing, aerospace, and power engineering, to name a few. As a consequence, systems have grown in complexity, which ultimately makes their design and operation more challenging.

Optimal operation of engineering systems often translates into setting discrete and/or continuous decision variables so as to maximize some performance index subject to feasibility constraints. This leads to the formulation of a mathematical program, [1], which becomes intractable in case of a large-scale system involving discrete decision variables due to the combinatorial explosion of the computational effort. One then has to sacrifice optimality for computational tractability and head for a resolution method that provides a feasible solution with, possibly, some performance guarantees.

In this thesis, we focus on those decision making problems that can be formulated as a large-scale Mixed Integer Linear Program (MILP).

MILPs arise as a natural framework when addressing problems of various nature (identification, verification, reachability analysis, and control) for

the class of Mixed Logical Dynamical (MLD) systems, originally introduced in [2]. MLD systems have been extensively studied in the systems and control literature [3, 4, 5, 6, 7, 8], because of their modeling capabilities that make them suited to represent various systems comprising both logical and physical components.

Several algorithms for solving generic MILPs have been developed over the years [9, 10, 11, 12, 11, 13], but the reduction of their computational complexity is still an open challenge, especially in case of large-scale problems. Recent work in [14] and [15] partially addresses this issue by introducing effective methods for solving large-scale MILPs with a constraint-coupled multi-agent structure where each agent is optimizing a subset of the decision variables with respect to its own cost function subject to its local constraints and additional global ones that are coupling its decision with that of the other agents. The original MILP is then reduced to multiple lower-dimensional MILPs, one per agent, that are repeatedly solved while exchanging information on the coupling. If the number of discrete decision variables per agent is small, then, computational tractability is recovered. In particular, by relying on the interconnected structure of the multi-agent system, the decentralized iterative scheme in [15] guarantees convergence to a feasible solution in a finite number of iterations and provides also a characterization of its sub-optimality level with respect to a centralized solution where the sum of the cost functions is minimized subject to all local and global constraints. The method is most effective if the number of coupling constraints is small with respect to the number of agents.

The aim of this thesis is to devise a procedure that manipulates a large-scale not necessarily structured MILP so as to highlight its partially decomposable structure, thus making it more suitable for the application of the resolution scheme in [15]. This calls for a method to rewrite the original program as an equivalent constraint-coupled multi-agent MILP by properly permuting the matrix defining the linear constraints into a

block-angular structure, where the blocks define both local constraints and local decision variables of the fictitious agents, and the border corresponds to the coupling constraints. Since the cost function is linear, it can always be decomposed into a sum of local cost functions according to the decision variable partition. The most convenient structure for the application of the algorithm in [15] needs to be enforced in the resulting matrix, which should have a large number of blocks and a thin border, with the discrete decision variables evenly divided among the agents associated to the blocks.

1.2 Contribution by chapter

The remainder of the thesis is organised as follows:

Chapter 2 provides some insights on MILP as a framework for optimal decision making, with reference to the class of MLD systems and energy systems as a possible application context.

Chapter 3 presents an overview on solution algorithms for MILPs with a special focus on well-established decomposition-based approaches. The chapter ends with the description of recently developed decentralized optimisation algorithms for constraint-coupled multi-agent MILPs.

Chapter 4 proposes a resolution scheme for a large-scale MILP that rests on its reformulation as a constraint-coupled multi-agent MILP and its decentralized solution. The MILP reformulation is based on the permutation of the constraint matrix to a block-angular form. A novel strategy based on a graph interpretation of the constraint matrix is presented and compared with alternative state-of-the-art algorithms for matrix manipulation. The proposed procedure is specifically tuned to work well with the adopted decentralized resolution method.

Chapter 5 focuses on performance evaluation of the proposed strategy for reducing a large-scale MILP to a constraint-coupled multi-agent

MILP. Implementation choices are described and motivated. An analysis of how different tuning parameters of the procedure affect the final result is also provided.

Chapter 6 concludes the thesis with some final remarks and suggestions for future developments.

Chapter 2

MILPs for optimal decision making

2.1 Why MILPs?

Mixed integer linear optimization deals with decision-making problems in which values for both continuous and discrete variables must be set so as to optimize a linear performance index subject to linear constraints. A compact *canonical form* for a generic Mixed Integer Linear Program (MILP) is the following:

$$\min_x \tilde{c}^\top x \tag{2.1a}$$

subject to:

$$\tilde{A}x \leq \tilde{b} \tag{2.1b}$$

$$x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} \tag{2.1c}$$

where vector x contains n_c continuous decision variables and n_d discrete ones. Vector $\tilde{c} \in \mathbb{R}^{n_c+n_d}$ defines the cost function, whereas matrix $\tilde{A} \in \mathbb{R}^{q \times n_c+n_d}$ and vector $\tilde{b} \in \mathbb{R}^q$ define q scalar constraints.

A decision vector x is *feasible* if it belongs to the mixed integer feasible

set

$$S = \{x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} : \tilde{A}x \leq \tilde{b}\}. \tag{2.2}$$

A decision vector $x \in S$ is an optimal solution to (2.1) if it minimizes the cost function $\tilde{c}^\top x$ over the feasibility set S. An optimal solution will be denoted in the sequel as $x^\star$.

MILPs arise in various contexts, from finance and production planning to any engineering application involving systems comprising interconnected continuous (such as temperature, power, mass, etc.) and/or discrete variables (on/off conditions, operating levels, etc.) that can be modeled as *Mixed Logical Dynamical* (MLD) systems. MLD systems have wide modeling capabilities and can describe the behavior of processes involving interleaved physical laws and logical rules subject to operating constraints. They include linear hybrid systems, finite state machines, and piecewise linear, possibly constrained, dynamical systems, [2].

An MLD system is described by the following set of linear equations:

$$s(t+1) = A_t s(t) + B_{1t} u(t) + B_{2t} \eta(t) + B_{3t} z(t) \tag{2.3a}$$

$$y(t) = C_t s(t) + D_{1t} u(t) + D_{2t} \eta(t) + D_{3t} z(t) \tag{2.3b}$$

$$E_{2t} \eta(t) + E_{3t} z(t) \leq E_{1t} u(t) + E_{4t} s(t) + E_{5t} \tag{2.3c}$$

where

- $s = \begin{bmatrix} s_c \\ s_l \end{bmatrix}$, $s_c \in \mathbb{R}^{q_c}, s_l \in \{0,1\}^{q_l}$, $q := q_c + q_l$
- $y = \begin{bmatrix} y_c \\ y_l \end{bmatrix}$, $y_c \in \mathbb{R}^{p_c}, y_l \in \{0,1\}^{p_l}$, $p := p_c + p_l$
- $u = \begin{bmatrix} u_c \\ u_l \end{bmatrix}$, $u_c \in \mathbb{R}^{m_c}, u_l \in \{0,1\}^{m_l}$, $m := m_c + m_l$

are the state, output and input vectors respectively, having both continuous and binary components, while

- $\eta \in \{0,1\}^{r_l}$

- $z \in \mathbb{R}^{r_c}$

are additional auxiliary variables.

In [2], well-posedness of the system of equations (2.3a)-(2.3c) defining the state $s(t+1)$ and the output $y(t)$ of a MLD system given the initial state $s(0)$ and the input u up to time t is discussed. Notions of output and state trajectory, equilibrium and stability are extended to the MLD class and optimal control is illustrated. Model predictive control (MPC) strategies are also discussed. In MPC, a finite-horizon constrained optimization problem needs to be solved at each time instant. Due to the hybrid nature of the variables and operational constraints involved, when addressing MPC with linear cost function and operating constraints for an MLD system, a MILP comes into play. Its size depends not only of the order of the system, but also on the time horizon length of the finite-horizon optimization problem, which can make it challenging to find a solution within one time step in case of fast processes.

An example of a MILP for the optimal operation of an MLD system over a finite-horizon is described next with reference to the power grid application context. The example clearly shows the potential of the MLD modeling framework and the natural connection of the related optimal control problems with mixed integer optimization. It is also instrumental to introduce the class of multi-agent MILPs that typically arise when modelling an aggregate of subsystems (agents), where each agent has its own set of decision variables and local constraints but all agents decisions are coupled due to the presence of a common resource.

2.2 A significant example

In this section we show an application of the MILP framework to energy systems and, in particular, to the problem of coordinating multiple

prosumers so as to provide balancing services to the grid. The example is taken from [16]. Here, we work out explicitly the MLD model of the multiple prosumers system before deriving the MILP formulation of the coordination problem. Note that integer variables will be coded as binary variables to match the MLD model (2.3).

We consider an Energy Service Provider (ESP) who acts as an aggregator of a pool of prosumers. The power exchanged by the pool with the grid is scheduled according to some reference daily profile established with the Transmission System Operator (TSO). However, the TSO can submit a request to the ESP to change the reference profile of a given amount during a time slot of a certain duration, so as to balance production and consumption at the grid level. The ESP has then to decide how to distribute the requested change among all prosumers in the pool so as to satisfy the TSO request up to a certain tolerance, while minimizing the operating costs.

The problem can be addressed by deriving a MLD model for the aggregated system and solving the resulting MILP as described next.

Consider a pool of m prosumers (agents), each one equipped with:

- a controllable generator G producing power $P_i^G > 0$;
- a programmable load L requiring power $P_i^L > 0$
- a battery storage device B, with exchange rate P_i^B such that $P_i^B < 0$ when charging and $P_i^B > 0$ when discharging.
- a known reference daily power exchange profile $\widetilde{P}_i$.

The power request of the pool is then given by

$$P = \sum_{i=1}^{m} (P_i^G + P_i^B - P_i^L).$$

The scheduling operation can be carried out dividing the one-day time horizon into M time slots of duration Δt_s. In the sequel, we shall denote

with t, $t \in \{0, 1, \ldots, M-1\}$, the time slot corresponding to $[t\Delta t_s, (t+1)\Delta t_s)$ and refer to the average values per time slot of the continuous variables.

Assume that a power variation request is submitted to the ESP with reference to the time slots $t_0, \ldots, t_f$.This request can be expressed as a variation in the power profile of the type: $\Delta P(t) \pm \epsilon \Delta P(t)$, with $\epsilon \in (0, 1)$ relative tolerance parameter. The ESP is, hence, expected to reschedule the operations of the different prosumers (their programmable loads, controllable generators, batteries) so as to meet the conditions:

$$(1-\epsilon)\,\Delta P(t) \leq P(t) - \widetilde{P}(t) \leq (1+\epsilon)\,\Delta P(t), \quad t = t_0, \ldots, t_f \tag{2.4}$$

$$P_i(t) = \widetilde{P}_i(t), \quad t = t_f + 1, \ldots, M-1, \tag{2.5}$$

while accounting for the additional operational constraints of the overall system (described in detail next).

$\widetilde{P}_i = \widetilde{P}_i^G + \widetilde{P}_i^B - \widetilde{P}_i^L$ denotes the pre-agreed reference power profile of prosumers i and depends on the the reference power profiles of the devices (generator $\widetilde{P}_i^G$, battery $\widetilde{P}_i^B$ and load $\widetilde{P}_i^L$), while $\widetilde{P} = \sum_{i=1}^m \widetilde{P}_i$. Note that (2.5) ensures that each prosumer maintains the original profile after the satisfaction of the request, avoiding rebound effects.

The programmable load of each prosumer is assumed to operate only at specific levels, i.e., , $P_i^L(t) \in \left\{0, \frac{\overline{P}_i^L}{n_i^L}, 2\frac{\overline{P}_i^L}{n_i^L}, \ldots, \overline{P}_i^L\right\}$. We shall assume in the following that $n_i^L = 2^{J_i^L} - 1$ with $J_i^L \in \mathbb{N}$. We can then provide a binary code for the admissible values of P_i^L through the binary variables $\delta_{i,j}^L(t)$, $j \in \{1, \ldots, J_L^i\}$, as follows:

$$P_i^L(t) = \frac{\overline{P}_i^L}{n_i^L} \sum_{j=1}^{J_i^L} \left(2^{j-1} \cdot \delta_{i,j}^L(t)\right) = \sigma_i^\top \delta_i^L(t) \tag{2.6}$$

where we set

$$\sigma_i = \frac{\overline{P}_i^L}{n_i^L} \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 2^{J_i^L-1} \end{bmatrix} \qquad \delta_i^L(t) = \begin{bmatrix} \delta_{i,1}^L(t) \\ \vdots \\ \delta_{i,J_i^L}^L(t) \end{bmatrix}.$$

If n_i^L cannot be expressed as $n_i^L = 2^{J_i^L} - 1$, then, $J_i^L = \left\lceil \log_2(n_i^L + 1) \right\rceil$ and adequate constraints have to be introduced to exclude the values of $P_i^L(t)$ in (2.6) that are not admissible.

In order to derive a MLD model of the aggregated prosumers system, we need to introduce the following additional variables:

- the state of charge $S_i(t)$ of the battery storage unit of prosumer i at the beginning of the time-slot t
- a binary variable $\delta_i^G(t)$ modeling the status of the generator of prosumer i during time-slot t (1 if it is on, 0 otherwise).

The resulting MLD is then characterized by:

- a **state vector** $s(t)$, collecting the state of the m battery storages at the beginning of the time-slot t:

$$s(t) = \begin{bmatrix} S_1(t) & \cdots & S_m(t) \end{bmatrix}^\top$$

- an **input vector** $u(t)$, collecting the values of all the variables that the ESP can modify in the time slot t:

$$u(t) = \begin{bmatrix} u_1(t) & \cdots & u_m(t) \end{bmatrix}^\top$$

where

$$u_i(t) = \begin{bmatrix} u_{i,c}(t) \\ u_{i,d}(t) \end{bmatrix} = \begin{bmatrix} P_i^G(t) \\ P_i^B(t) \\ \delta_i^G(t) \\ \delta_i^L(t) \end{bmatrix}$$

The evolution in time of the aggregated system is subject to operating constraints related to each time slot $t = t_0, \ldots, M-1$ and specified for all prosumers $i = 1, \ldots, m$:

Battery storage dynamics

$$S_i(t+1) = S_i(t) - \Delta t_s P_i^B(t) \tag{2.7a}$$

$$= S_i(0) - \Delta t_s \sum_{s=0}^{t} P_i^B(s) \tag{2.7b}$$

Min/Max energy level

$$\underline{S}_i \leq S_i(t) \leq \overline{S}_i \tag{2.8}$$

Discharging/Charging rates limitation

$$P_i^{B,c} \leq P_i^B(t) \leq P_i^{B,d} \tag{2.9}$$

Min/Max power produced by G

$$\delta_i^G(t)\underline{P}_i^G \leq P_i^G(t) \leq \delta_i^G(t)\overline{P}_i^G \tag{2.10}$$

Energy consumption by L

$$\sum_{t=t_0}^{M-1} \Delta t_s P_i^L(t) = \sum_{t=t_0}^{M-1} \Delta t_s \sigma_i^\top \delta_i^L(t) = E_i^L \tag{2.11}$$

where E_i^L is the amount of energy required by the load and constraint (2.11) makes sure the load receive all the required energy.

The standard form (2.3) can be derived by suitably expressing the above relationships in terms of the introduced state and input variables.

In particular, the state equation (2.7) can be brought in the form (2.3a) as follows:

$$s(t{+}1) = \begin{bmatrix} S_1(t+1) \\ \vdots \\ S_m(t+1) \end{bmatrix} = s(t) - \underbrace{\Delta t_s \begin{bmatrix} v_1^s & 0_{1\times J_1^L+3} & \cdots & 0_{1\times J_m^L+3} \\ 0_{1\times J_1^L+3} & v_2^s & \cdots & 0_{1\times J_m^L+3} \\ \vdots & \ddots & \ddots & \vdots \\ 0_{1\times J_1^L+3} & 0_{1\times J_2^L+3} & \cdots & v_m^s \end{bmatrix}}_{B_{1t}} \begin{bmatrix} u_1(t) \\ \vdots \\ u_m(t) \end{bmatrix}$$

where $v_i^s = \begin{bmatrix} 0 & 1 & 0 & 0_{1\times J_i^L} \end{bmatrix}$.

As for constraints (2.7) - (2.11), they can be re-arranged in the form (2.3c) as shown in Appendix A. We next focus on the formulation of the rescheduling problem that has to be solved for satisfying the TSO request over time-slots $t_0, \ldots, t_f$.

The following additional constraints need to be introduced:

Flexibility limitation of L

$$P_i^L(t) = \sigma_i^\top \delta_i^L(t) = \widetilde{P}_i^L(t), \quad t < t_i^{L,0} \vee t > t_i^{L,f} \tag{2.12}$$

where $t_i^{L,0}$ and $t_i^{L,f}$ respectively denote the first and the last time slots in which the load can be modulated or shifted.

Power variation required by the TSO

$$(1-\epsilon)\Delta P(t) \leq \sum_{i=1}^{m} \left(P_i^G(t) + P_i^B(t) - \sigma_i^\top \delta_i^L(t)\right) - \widetilde{P}(t) \leq (1+\epsilon)\Delta P(t), \tag{2.13}$$

for $t = t_0, \ldots, t_f$

Rebound-Effect avoidance

$$P_i(t) = P_i^G(t) + P_i^B(t) - \sigma_i^\top \delta_i^L(t) = \widetilde{P}_i(t), \quad t = t_f + 1, \ldots, M-1 \tag{2.14}$$

where the last two are (2.4) and (2.5) rewritten here in terms of the input variables.

Among the different admissible solutions to the problem, we favor the one minimizing the overall operating cost:

$$J(\cdot) = \sum_{i=1}^{m} \sum_{t=t_0}^{M-1} \left(C_i^G P_i^G(t) + C_i^B \left|P_i^B(t) - P_i^B(t-1)\right| + C_i^L \left|P_i^L(t) - \widetilde{P}_i^L(t)\right|\right),$$

where $C_i^G > 0$ is the cost of producing one unit of power with the controllable generator, $C_i^B > 0$ is the cost associated to the aging of the battery, and $C_i^L > 0$ is the cost that penalises changes in the programmable load consumption profile with respect to its original schedule.

By replacing the absolute values appearing in the cost function above with the auxiliary decision variables $h_i^B(t)$, $h_i^L(t)$, $i = 1, \ldots, m$:

$$J(\cdot) = \sum_{i=1}^{m} \sum_{t=t_0}^{M-1} \left(C_i^G P_i^G(t) + C_i^B h_i^B(t) + C_i^L h_i^L(t)\right) \tag{2.15}$$

and imposing the following constraints

$$P_i^B(t) - P_i^B(t-1) \leq h_i^B(t), \qquad t = t_0, \dots, M-1 \tag{2.16a}$$
$$-P_i^B(t) + P_i^B(t-1) \leq h_i^B(t), \qquad t = t_0, \dots, M-1 \tag{2.16b}$$
$$\sigma_i^\top \delta_i^L(t) - \tilde{P}_i^L(t) \leq h_i^L(t), \qquad t = t_0, \dots, M-1 \tag{2.16c}$$
$$-\sigma_i^\top \delta_i^L(t) + \tilde{P}_i^L(t) \leq h_i^L(t), \qquad t = t_0, \dots, M-1 \tag{2.16d}$$

where $P_i^L(t)$ has been replaced ith $\sigma_i^\top \delta_i^L(t)$ according to (2.6), the MILP framework is recovered.

The ESP has in fact to determine the values for the control input and the auxiliary decision variables along the time horizon $[t_0, M-1]$ so as to minimize (2.15) subject to the MLD system constraints (2.7)-(2.11), the operating constraints (2.12)-(2.14) due to the TSO request, and the constraints (2.16) on the introduced auxiliary decision variables.

Separability of (2.15) across i allows to express it as a sum of contributions from different prosumers. Indeed, all variables referred to the same prosumer i in $t = t_0, \dots, M-1$ can be collected in a vector x_i of *local variables*:

$$x_i = \begin{bmatrix} u_i(t_0)^\top & h_i^B(t_0) & h_i^L(t_0) & \cdots & u_i^\top(M-1) & h_i^B(M-1) & h_i^L(M-1) \end{bmatrix}^\top$$

so that (2.15) can be written as

$$J(\cdot) = \sum_{i=1}^{m} c_i^\top x_i$$

where $c_i^\top = \begin{bmatrix} c_i^u & C_i^b & C_i^L & \cdots & c_i^u & C_i^b & C_i^L \end{bmatrix}$ with $c_i^u = \begin{bmatrix} C_i^G & 0 & 0 & 0_{1\times J_i^L} \end{bmatrix}$

As a result, the demand response problem can be formulated as the

following constraint-coupled multi-agent MILP:

$$\min_{x_1,\dots,x_m} \sum_{i=1}^{m} c_i^\top x_i \tag{2.17a}$$

subject to:

$$(2.13), \quad t = t_0, \dots, t_f \tag{2.17b}$$

$$\begin{cases} (2.7) - (2.10), (2.16),\ t = t_0, \dots, M-1 \\ (2.11) \\ (2.12),\ t = t_0, \dots, t_i^{L,0}, t_i^{L,f}, \dots, M-1 \\ (2.14),\ t = t_f + 1, \dots, M-1 \end{cases} \qquad i = 1, \dots, m \tag{2.17c}$$

where

- x_i is the *local decision vector* of prosumer i and contains both binary and continuous variables

- (2.7) - (2.12), (2.14), (2.16) are linear constraints that jointly define a *local constraint set* X_i of prosumer i

- (2.13) is the *coupling constraint*, arising from the need to coordinate all the prosumers in the pool so as to satisfy the TSO flexibility request. This constraint can be expressed in the form $\sum_{i=1}^{m} A_i x_i \leq b_i$ (see the derivations in the Appendix B).

Formally, the general structure of a constraint-coupled multi-agent MILP is as follows:

$$\min_{x_1,\dots,x_m} \sum_{i=1}^{m} c_i^\top x_i \tag{2.18a}$$

subject to:

$$\sum_{i=1}^{m} A_i x_i \leq b \tag{2.18b}$$

$$x_i \in X_i,\ i = 1, \dots, m, \tag{2.18c}$$

where X_i in (2.18c) is the mixed-integer set defined by the local constraints and compactly described as:

$$X_i = \{x_i \in \mathbb{R}^{n_{c,i}} \times \mathbb{Z}^{n_{d,i}} : D_i x_i \leq d_i\}, \tag{2.19}$$

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 (2.18b) is defined by vector $b \in \mathbb{R}^p$ and matrix $A_i \in \mathbb{R}^{p \times n_i}$ where $n_i = n_{c,i} + n_{d,i}$ is the total number of decision variables of agent i.

Problems in the form (2.18) are not confined to power systems control, but arise in several contexts involving resources allocation and management: from traditional combinatorial challenges, as the knapsack problem [17], to finance [18], and network utility optimization [19].

Chapter 3

Resolution schemes for large-scale MILPs

3.1 Optimal approaches

Despite the simplicity of their formulation, which involves linear cost and constraints, large-scale MILPs are difficult to solve, because of their intrinsic combinatorial complexity due to the presence of integer decision variables.

In principle, an optimal solution to a MILP can be found combining the traditional *simplex method* [20] with the enumeration of all possible combinations of values for the n_d discrete components of x. Indeed, each combination defines a different LP, whose optimal solution provides values for the n_c continuous components of vector $x^\star$. Although possible in theory, the number of combinations to test grows exponentially with n_d, thus making such a strategy inefficient and often not viable in practice. To have an idea, consider the MILP provided in the energy application example in Section 2.2 formulated for a pool of $m = 10$ prosumers, each having only two levels for the load (on-off state associated to only one binary variable $\delta_i^L(t)$). The corresponding formulation, setting $t_0 = 0$ and considering a 24-hours time horizon divided in $M - t_0 = M = 96$ time slots

of 15-minutes duration, has a total of $n_c = M \cdot 4 \cdot m = 3840$ continuous and $n_d = 2 \cdot M \cdot m = 1920$ binary variables. With explicit enumeration, even assuming an unrealistic solving time of 10^{-10} seconds for each of the 2^{n_d} LPs, it would take $3.0115 \cdot 10^{560}$ years to find an optimal solution via explicit enumeration.

3.1.1 Branch and bound algorithms

A widely adopted algorithmic principle that allows to overcome such limitation is *Branch and Bound* (B&B). Devised in 1960 by Land and Doig in [13] and embedded in state-of-the-art software for integer programming, B&B is a search strategy based on a binary enumeration tree. Two operations are at the core of the procedure: *branching* that recursively divides the feasible set S into subsets, and *bounding*, that attempts to prune the enumeration exploiting the optimization sub-problems inducted by the branching. Different policies can be adopted for the two steps, resulting in different B&B strategies.

A common choice for the branching strategy consists in creating two disjoint sub-problems for each node of the tree by restricting the range of values that a given discrete variable can assume. This approach, known as *variable branching*, generates two *children* for each node by splitting the feasibility set S into two disjoint subsets and gives to the enumeration tree its binary structure. However, *branching* alone is just an enumeration technique. It is the *bounding* step that actually makes B&B competitive (and viable) with respect to exhaustive enumeration, because it allows to decide whether a branch can lead to the optimal solution (by further branching) or not, thus greatly reducing the number of combinations actually explored. There are many alternatives to bound the objective function value of a sub-problem. The simplest strategy is the *linear programming bounding* that provides a lower bound for the optimal objective function value $z^\star$ by relaxing the integrality constraints and solving the resulting LP.

Overall, a B&B procedure adopting the mentioned policies performs the

enumeration as follows. It starts by solving a LP relaxation of the original MILP (2.1) at the *root* node of the tree. If the LP is infeasible, enumeration stops since no solution for the original MILP can be found. If, instead, all the n_d originally discrete components of the LP solution $\tilde{x}^0 = (\tilde{x}^0_c, \tilde{x}^0_d)$ assume integer values, $\tilde{x}^0$ is also optimal for the original problem (2.1). Otherwise, the branching strategy selects one of the components in $\tilde{x}^0_d$ assuming a fractional value, say x_j, and generates two *child nodes* adding constraints

$$x_j \leq \left\lfloor \tilde{x}^0_j \right\rfloor \tag{3.1}$$

$$x_j \geq \left\lceil \tilde{x}^0_j \right\rceil \tag{3.2}$$

to the corresponding sub-problems. The procedure is repeated on the each child node h and depending on the optimal solution $\tilde{x}^h = (\tilde{x}^h_c, \tilde{x}^h_d)$ of the LP relaxation of the associated sub-problems the procedure can:

- *prune* the branch by *infeasibility* if the LP has no solution;
- *prune* the branch by *integrality* if all components in $\tilde{x}^h_d$ assume an integer value. The corresponding value $\tilde{z}^h$ of the LP objective function is an upper bound on the optimal solution $z^\star$ of the original MILP;
- *prune* the branch by *bound* if $\tilde{x}^h_d$ is not an integral vector and $\tilde{z}^h$ is greater then or equal to the best known upper bound on $z^\star$. This, indeed, ensures that no better solution can be found proceeding along the branch and allows to implicitly enumerate all its nodes;
- *branch* on a non-integer component of $\tilde{x}^h_d$ when the optimal solution of the LP sub-problem provides a value $\tilde{z}^h$ for the objective function that is lower than the best known upper bound.

The procedure is repeated until no open branch is left to explore.

Implicit enumeration of the discrete part of the feasibility set allows to retrieve a solution to (2.1) rapidly and without sacrificing its optimality. The policies chosen for the branching and bounding steps affect the speed

of the algorithm, thus becoming crucial as the size of the problem increases. Indeed, the strength of any B&B approach consists in the opportunity of pruning by bounding, which avoids explicit exploration of portions of S. In general, computational time reduces when the number of pruned branches increases or the level at which pruning occurs is higher. Depending on the specific application, one may generate bounds solving different relaxations of the sub-problems such as the Lagrangian relaxation, [21], described in Section 3.2.1.

3.1.2 Cutting plane method

In general, solving to optimality the LP relaxation of (2.1) provides a solution $x^{\star}_{LP}$ that minimizes the objective function value but does not necessarily satisfy the neglected integrality constraints. However, it might happen that $x^{\star}_{LP}$ satisfies the integrality constraints, then, it is optimal for the original MILP. In order to make this more likely to happen, one can add to the original MILP linear constraints that reduces the feasibility region of its LP relaxation while preserving all admissible values in the mixed-integer feasibility set S.

In principle, one should add all the inequalities that describe the smallest polyhedron containing all admissible points for (2.1). Such polyhedron, called *convex hull* of S and denoted as $\text{conv}(S)$, defines the *ideal formulation* for (2.1) and plays a crucial role in the resolution of MILP problems. Indeed, solving the LP relaxation of a problem (2.1) defined over the convex hull of its feasibility set allows to retrieve a solution $x^{\star}_{LP}$ that coincides with the optimal one of (2.1).

A possible resolution strategy may, then, add a-priori inequalities to the original formulation in the attempt of retrieving a description of the convex hull of its feasibility set. Unfortunately, finding the description of $\text{conv}(S)$ is, in general, not trivial and requires adding a considerable number of constraints to the linear relaxation, thus making its solution more challenging.

For this reason, in 1958, Ralph Gomory proposed an alternative approach known as *cutting plane method* [22], which is based on the recursive generation of *cuts* to provide a local description of the convex hull. A *cut* is an inequality constraint that tightens the solution space of the relaxed problem without cutting out any admissible solution in S. The main idea behind Gomory's algorithm is to exploit cuts to build a local description of the convex hull of S only in a neighbourhood of the optimal solution $x^\star$. However, since such solution is not known a priory the characterisation of $\text{conv}(S)$ is obtained through an iterative procedure. At each step of the procedure, the LP relaxation of the current formulation is solved to optimality. If its solution satisfies the integrality constraints, then it is also optimal for the original problem and the procedure stops. Otherwise, one or more cuts are added to the formulation solving a *separation problem*, aimed at excluding the current fractional solution from the feasibility set of the LP relaxation while preserving all the feasible solutions of the original problem.

In principle, the cutting planes method can find an optimal solution in a finite (possibly high) number of iterations. However, its efficiency in tightening the feasibility region of the LP relaxation depends on the separation problem and the computational effort required to solve it, which may vary depending on the problem at hand. For this reason the generation of cuts is often integrated in a B&B scheme, resulting in the so-called *Branch and Cut* approach (see [11] and [23]). This leads to a B&B approach where one or more cutting-plane steps are performed on the sub-problems before the branching operation. The resulting algorithm consists in a standard B&B where one can decide, for each *un-pruned* branch, whether to tighten the solution space of the sub-problem before generating the child nodes. This allows to produce potentially tighter bounds, and increase the pruning opportunities, thus reducing the overall computational time of the B&B procedure.

3.1.3 Column generation

Column generation [24] is an iterative technique used to solve linear programming problems having a large number of decision variables, possibly exponential in the number of constraints, for which direct resolution is typically not viable or extremely heavy from a computational point of view.

At each iteration, the algorithm considers only a subset of decision variables and solves a restricted version of the original problem defined over such variables only. The original problem is usually called *master problem (MP)* whilst the version defined over the subset of variables is the *restricted master problem (RMP)*. Once the RMP is solved, the algorithm determines whether the solution found is also optimal for the whole problem or if some variables should be added to the formulation to improve the result, together with the corresponding column in the constraint matrix. This decision is made solving a *pricing sub-problem* that starts from the current RMP solution and searches for a non-basic variable with negative reduced cost. The procedure stops when no variable can be added to improve the solution of the current restricted problem, which is thus optimal also for the original MILP.

Notice that the column generation algorithm provides solution for linear programs, but is not a resolution scheme for MILP problems *per sé*. Nevertheless, it proves to be extremely efficient in the *Branch and Price* strategies, where column generation is applied at each node to solve the linear relaxation of the associated sub-problem.

3.2 Heuristic approaches

As the size of the instance increases, the computational effort required to solve a MILP at optimality may become prohibitive. One may then try to take advantage of the special structure of the decision-making problem formulation, if any, to obtained simplified reformulations. Such

reformulations can either replace the standard linear relaxations in a B&B approach - to retrieve an optimal solution - or they can be combined with suitable problem-specific heuristics to recover a feasible (but possibly sub-optimal) solution, see e.g. [25] and [26].

3.2.1 Lagrangian relaxation

A first and well known reformulation is the *Lagrangian relaxation*. It applies to all problems in the form (2.1) when the constraints (2.1b) can be partition into two subsets, namely $\widetilde{A}^1 x \leq \tilde{b}^1$ and $\widetilde{A}^2 \leq \tilde{b}^2$, such that the reformulation obtained by removing one of the subset of constraints, say $\widetilde{A}^1 x \leq \tilde{b}^1$, called *complicating constraints*, can be more efficiently solved with respect to the original problem.

More precisely, consider the MILP

$$\min_x \tilde{c}^\top x \tag{3.3a}$$

subject to:

$$\widetilde{A}^1 x \leq \tilde{b}^1 \tag{3.3b}$$

$$\widetilde{A}^2 x \leq \tilde{b}^2 \tag{3.3c}$$

$$x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} \tag{3.3d}$$

Its Lagrangian relaxation (LR) can be obtained by removing the complicating constraints (3.3b) from the formulation and adding a penalisation of their violation in the cost function:

$$\min_x \tilde{c}^\top x - \lambda \left(\tilde{b}^1 - \tilde{A}^1 x\right) \tag{3.4a}$$

subject to:

$$\tilde{A}^2 x \leq \tilde{b}^2 \tag{3.4b}$$

$$x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} \tag{3.4c}$$

Vector $\lambda \geq 0$ collects the *Lagrange multipliers* and determines the impact of the constraints violation in the objective function. The choice of λ is, in

general, not unique but it strongly affects the quality of the lower bound on the optimal cost of the original problem provided by its Lagrangian relaxation. The tightest bound can be obtained solving the *Lagrangian dual*:

$$\max_{\lambda \geq 0} \left\{ q(\lambda) = \min_{x \in Q} \left\{ \tilde{c}^\top x - \lambda \left(\tilde{b}^1 - \widetilde{A}^1 x \right) \right\} \right\} \tag{3.5}$$

where the set Q is equal to:

$$Q = \{x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} : \widetilde{A}^2 x \leq \tilde{b}^2\} \tag{3.6}$$

Since the *dual function* $q(\lambda)$ is concave and non-differentiable, the dual problem (3.5) is usually solved by means of the *subgradient algorithm*. The lower-bound obtained by the joint solution of (3.5) and (3.4) can be proved to be at least as tight as the one provided by a linear relaxation, and typically better, [27, Corollary 8.4].

Therefore, whenever the constraints of the original problem can be partitioned as described, a Lagrangian reformulation can be used in place of an LP relaxation in B&B algorithms to improve the performance, since:

- it leads to a simplification of the optimization problem that is solved at each node of the enumeration tree;
- it typically increases the pruning opportunities.

Dantzig-Wolfe relaxation

The Dantzig-Wolfe reformulation (DWR) allows to retrieve an alternative relaxation for MILPs in the form (3.3), where (3.3b) are still assumed to be complicating constraints, while characterising set conv(Q), with $Q = \{x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d} : \widetilde{A}^2 x \leq \tilde{b}^2\}$ is *"easy"*. In particular, the DWR consists in rewriting the original problem by substituting constraints (3.3c)-(3.3d) with the condition $x \in \text{conv}(Q)$.

Such constraint is, then, implicitly enforced expressing x as the sum of a convex combination of the vertices $\{v^k\}_{k=1}^K$ and a conic combination of the extreme rays $\{r^h\}_{h=1}^H$ of conv(Q):

$$x = \sum_{k=1}^{K} \left(\ell_k v^k\right) + \sum_{h=1}^{H} \left(\mu_h r^h\right) \tag{3.7}$$

where $\ell \geq 0$ collects the convex combination weights $\{\ell_k\}_{k=1}^K$, which are such that $\sum_{k=1}^K \ell_k = 1$, whilst $\mu \geq 0$ collects the conic combination coefficients $\{\mu_h\}_{h=1}^H$. The obtained relaxation

$$\min_x \ \tilde{c}^\top \left(\sum_{k=1}^{K} \left(\ell_k v^k\right) + \sum_{h=1}^{H} \left(\mu_h r^h\right)\right) \tag{3.8a}$$

subject to:

$$\tilde{A}^1 \left(\sum_{k=1}^{K} \left(\ell_l v^k\right) + \sum_{h=1}^{H} \left(\mu_h r^h\right)\right) \leq \tilde{b}^1 \tag{3.8b}$$

$$\sum_{k=1}^{K} \ell_k = 1 \tag{3.8c}$$

$$\ell \in \mathbb{R}_+^K, \mu \in \mathbb{R}_+^H \tag{3.8d}$$

is expressed with respect to the vectors of the weights of the combination ℓ and μ.

Note that, in general, problem (3.8) has a large number of decision variables, namely as many as the vertices and rays of set $\text{conv}(Q)$, and is, thus, solved via column generation.

3.3 Decentralised approaches for structured MILPs

Constraint-coupled multi-agents MILPs are an example of structured problem suited for the application of Lagrangian or Dantzig-Wolfe reformulation. Indeed, the set of constraints of a problem in the form (2.18) can be always partitioned into two subsets coinciding with coupling and local constraints and such that - neglecting the first ones - the problem becomes separable across the agents and is, hence, reduced to the solution of many lower-dimensional MILPs.

Solving to optimality any of the two relaxations, however, does not necessarily provide a feasible solution for (2.18). In particular, the Dantzig-Wolfe reformulation substitutes the set defined by the local constraints with its convex hull, effectively solving the following constinuous linear problem:

$$\begin{aligned}
&\min_{x_1,\dots,x_m} \sum_{i=1}^{m} c_i^\top x_i \\
&\text{subject to:} \\
&\sum_{i=1}^{m} A_i x_i \leq b \\
&x_i \in \text{conv}(X_i),\ i = 1, \dots, m,
\end{aligned}$$

whose optimal solution $x^\star = [x_1^\star, \dots, x_m^\star]$ may not satisfy (2.18c) since

$$x_i^* \in \text{conv}(X_i) \implies x_i^* \in X_i. \tag{3.9}$$

The Lagrangian relaxation, instead, neglects the coupling constraints and makes the problem separable across the agents. Its solution can be retrieved with a two step procedure where the dual problem is solved first

$$\max_{\lambda \geq 0} -\lambda^\top b + \sum_{i=1}^{m} (c_i^\top + \lambda^\top A_i) x_i$$

and then the primal solution is recovered as

$$x_i(\lambda) \in \arg\min_{x_i \in X_i} (c_i^\top + \lambda^\top A_i) x_i \tag{3.10}$$

where λ is set equal to the solution $\lambda^\star$ of the dual. The obtained $x_i(\lambda^\star)$, $i = 1, \dots, m$ are, however, not guaranteed to jointly satisfy the coupling constraint and the standard recovery approaches of the convex case (see [28]), do not apply here. In the recent work [14], an approach is proposed that guarantees feasibility of the solution obtained via (3.10) when λ is set equal to the solution of the dual of a tightened version of the original

problem (2.18)

$$\min_{x_1,\dots,x_m} \sum_{i=1}^{m} c_i^\top x_i \tag{3.11}$$
$$\text{subject to:}$$
$$\sum_{i=1}^{m} A_i x_i \leq b - \rho$$
$$x_i \in X_i,\ i = 1,\dots,m$$

where the available amount of resource b is reduced of a suitable quantity $\rho > 0$.

Let us define vector $\tilde{\rho} \in \mathbb{R}^p$ with components

$$\tilde{\rho}_j = p \max_{i\in\{1,\dots,m\}} \left\{ \max_{x_i\in X_i} [A_i]_j x_i - \min_{x_i\in X_i} [A_i]_j x_i \right\},\ j = 1,\dots,p, \tag{3.12}$$

where $[A_i]_j$ denotes the j-th row of A_i and $\tilde{\rho}_j$ the j-th entry of $\tilde{\rho}$. Suppose that there exists a unique solution to the dual of the tightened problem (3.11) and to the associated LP when setting $\rho = \tilde{\rho}$. If we denote by $\lambda^\star_{\tilde{\rho}}$ the solution to the dual, then, in [14] it is shown that $x_i(\lambda^\star_{\tilde{\rho}})$, $i = 1,\dots,m$, (cf. equation (3.10)) is feasible for the original problem (2.18). Performance guarantees are also provided, with a gap that scales linearly with the infinity norm of the tightening vector. The approach is effective only if the number p of coupling constraints is small compared to the number m of agents.

In [15] the choice of the tightening vector ρ is performed in an adaptive fashion and integrated into a decentralized resolution scheme that aims at determining a feasible solution to (2.18) through a similar reasoning than [14].

The proposed algorithm is a variant of the dual sub-gradient algorithm and is reported below for ease of reference (cf. Algorithm 1). At each iteration, agents generate tentative primal solutions $x_i(k+1)$ (see step 7 in Algorithm 1) and share them with a coordination unit, which in turn update the value for the tightening ρ and the dual variable λ. Due to the

separable structure of the problem, dualization of the coupling constraints decomposes the original MILP into m sub-problems, one for each agent with reduced complexity and subject to local constraints only. Reduced conservativeness with respect to [14] is guaranteed by the update policy for the tightening. At each iteration the value $\rho(k)$ is refined based only on explored values of the primal solutions $x_i \in X_i$ for $i = 1, ..., m$ and not on the overall set X_i (see step 12).

In [15], Algorithm 1 with ḫoice of $\{\alpha(k)\}_{k\geq 0}$ such that $\lim_{k\to\infty} \alpha(k) = 0$ and $\sum_{k=0}^{\infty} \alpha(k) = \infty$ is shown to converge in a finite number of iterations to a feasible solution with a final value for the tightening vector ρ that is no worse (and typically better) than the worst-case $\tilde{\rho}$ defined in (3.12). This leads to a better performing solution and a less conservative approach which can be applied to a wider class of problems than the approach in [14].

Possible variants of Algorithm 1 can be devised to improve the quality of the solution, while preserving its finite convergence properties.

Algorithm 1 Decentralised Multi-Agent MILP

$\lambda(0) = 0$;

$\overline{s}_i(0) = -\infty, \qquad i = 1, \dots, m$;

$\underline{s}_i(0) = +\infty, \qquad i = 1, \dots, m$;

$k = 0$

repeat

  for $i = 1$ to m **do**

    $x_i(k+1) \leftarrow \arg \min_{x_i \in \mathrm{vert}(X_i)} (c_i^\top + \lambda(k)^\top A_i) x_i$

  end for

  $\overline{s}_i(k+1) = \max\{\overline{s}_i(k), A_i x_i(k+1)\}, \qquad i = 1, \dots, m$

  $\underline{s}_i(k+1) = \max\{\underline{s}_i(k), A_i x_i(k+1)\}, \qquad i = 1, \dots, m$

  $\rho_i(k+1) = \overline{s}_i(k+1) - \underline{s}_i(k+1), \qquad i = 1, \dots, m$

  $\rho(k+1) = p \max_{i=1,\dots,m} \{\rho_i(k+1)\}$

  $\lambda(k+1) = \Big[\lambda(k) + \alpha(k)\Big(\sum_{i=1}^{m} A_i x_i(k+1) - b + \rho(k+1)\Big)\Big]_+$

  $k \leftarrow k + 1$

until a stopping criterion is met

Chapter 4

Large-scale MILPs with hidden multi-agent structure

4.1 Proposed resolution scheme

The presence of a large number of discrete variables is the major obstacle to the solution of a MILP due to the exponential complexity growth. This issue has been partially tackled in the literature for structured problems, and, in particular, for constraint-coupled multi-agent MILPs. Section 3.3 shows how a feasible solution to constraint-coupled multi-agent MILPs with performance guarantees can be obtained through a decentralized scheme where the original MILP is decomposed into multiple lower-dimensional MILPs and a coordination layer is introduced to impose the coupling constraint, thus providing a scalable resolution strategy. However, this method is tailored to the specific structure of the program and is hence not directly applicable to a general (large-scale) MILP.

In this chapter we propose an automatic procedure aiming at reformulating a generic large-scale MILP to highlight its hidden multi-agent structure. By combining such a procedure with the decentralized resolution scheme in [15], we shall provide a general-purpose tool to large-scale MILP optimization.

4.2 Problem statement

The constraint-coupled multi-agent MILP (2.18), with local constraint sets defined in (2.18c), can be rewritten in the general MILP form (2.1) by setting $\tilde{c}^\top = [c_1^\top \cdots c_m^\top]$ and by collecting all local and global constraints in the inequality

$$\underbrace{\begin{bmatrix} D_1 & 0 & \dots & 0 \\ 0 & D_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & D_m \\ A_1 & A_2 & \dots & A_m \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_m \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \\ b \end{bmatrix}}_{\tilde{b}} \tag{4.1}$$

where the block matrix $\tilde{A}$ is characterized by a *singly-bordered block-angular* structure, [29].

The problem of disclosing the hidden multi-agent structure of a MILP then amounts to finding a suitable transformation of matrix $\tilde{A}$ in (2.1) so as to bring it into the singly-bordered block-angular form in (4.1). Figure 4.1 provides a pictorial view of a (sparse) matrix $\tilde{A}$ (left plot) which, under suitable manipulations, is brought into a singly-bordered block-angular form (right plot). This allows to compute a feasible solution of the generic MILP via the iterative procedure described in Section 3.3, where each block on the main diagonal of the transformed $\tilde{A}$ defines the agents' local variables and constraints, while the border at the bottom of the transformed $\tilde{A}$ corresponds to the coupling constraints.

Notice that

- in order to preserve the mixed-integer nature of the problem, only transformations involving permutations of rows and columns are adopted

- integer decision variables have to be evenly distributed among the agents to equally split the computational load

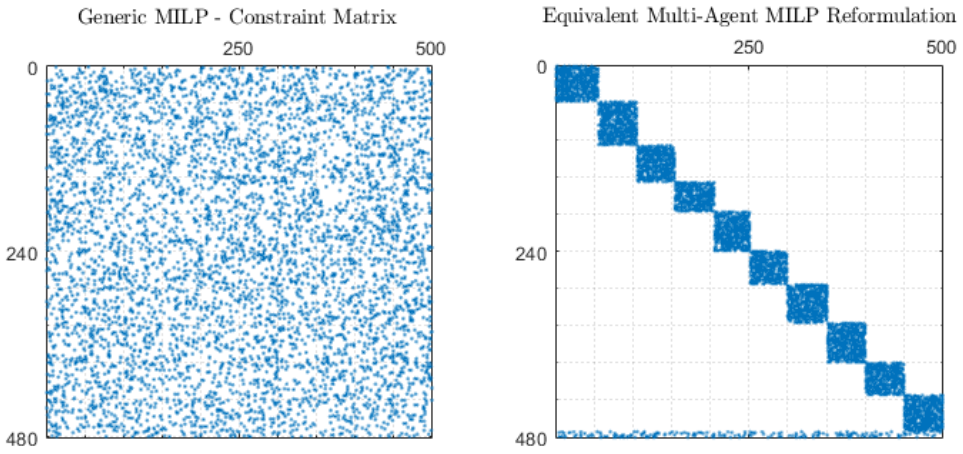


Figure 4.1: Transformation of a generic MILP constraint matrix (left) into its singly-bordered block-angular form (right), highlighting the presence of a constraint-coupled multi-agent structure.

- the border of the matrix corresponding to the coupling constraints has to be thin compared to the number of agents (blocks) for the decentralized resolution scheme to be effective.

4.2.1 Reformulation as a graph partitioning problem

The problem of retrieving a block-angular form for a (sparse) matrix $\tilde{A}$ is well-known in different fields. Numerical analysis techniques take advantage of the block-angular forms to parallelize computations in LU or QR decomposition, matrix multiplication and inversion. In optimization, it is adopted to efficiently solve large-scale LPs via Dantzig-Wolfe reformulation [30]. 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 $\tilde{A}$ is derived, then a partition of the hyper-graph nodes in m sets is performed. Finally, the hyper-graph

partition is re-interpreted as a permutation of the matrix $\tilde{A}$. Unfortunately, available algorithms for hyper-graph partitioning are not directly applicable to our setting, mainly because we need to balance the number of discrete decision variables among the blocks corresponding to the agents. The idea developed in this thesis is to introduce a novel partitioning algorithm that is inspired by those in the literature but meets our requirements. To this purpose, we first need to recall some notions of graph theory.

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, then, the standard notion of graph is recovered since each net would be equivalent to an edge connecting two nodes. The nodes in a net are called *pins* and the set of pins in a net n_i is denoted as $\text{Pins}(n_i)$. The set of nets connected to a node u_j is instead denoted as $\text{Nets}(u_j)$. A net n_i is said to be *incident* on a node u_j if $u_j \in \text{Pins}(n_i)$, and a node u_h is named a *neighbour* of u_j if there exists a net n_i incident on (i.e. connecting) both nodes.

Figure 4.2a shows an example of a hyper-graph $\mathcal{H}$. Each node u_j is represented with a circle containing the node label u_j, while each net n_i is depicted as a solid square labelled with n_i. A net i is incident on a node j if there exists a line connecting square n_i with circle u_j. For example, net n_1 is incident on nodes u_1, u_2 and u_3, that are its pins (i.e. $\text{Pins}(n_1) = \{u_1, u_2, u_3\}$, whilst nets n_2, n_3 and n_4 belong to $\text{Nets}(u_4)$.

An *m-way* node partition of a hyper-graph $\mathcal{H}$ is a collection $\Pi = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$ of subsets of nodes (each subset is called *part*) such that:

- $\mathcal{U}_h \subset \mathcal{U}, \mathcal{U}_h \neq \emptyset, \qquad h = 1, ..., m;$
- $\mathcal{U}_h \cap \mathcal{U}_l = \emptyset, \qquad 1 \leq h < l \leq m;$
- $\bigcup_{h=1}^{m} \mathcal{U}_h = \mathcal{U}.$

Figure 4.2b shows a possible 3-way partition $\Pi = \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3\}$ of the hyper-graph $\mathcal{H}$ depicted in Figure 4.2a with $\mathcal{U}_1 = \{u_1, u_2, u_3\}$, $\mathcal{U}_2 = \{u_5, u_6\}$, and

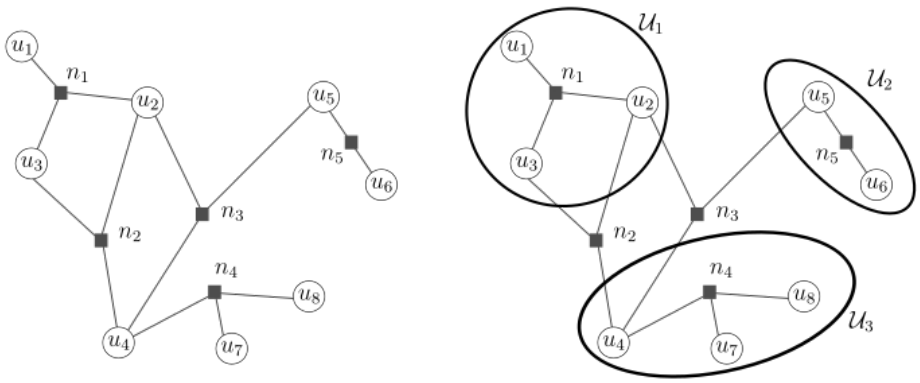


Figure 4.2: Example of hyper-graph $\mathcal{H}$ (a) and a possible 3-way partition (b).

$\mathcal{U}_3 = \{u_4, u_7, u_8\}$. 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. With reference to Figure 4.2b, n_1, n_4, and n_5 are internal nets while n_2 and n_3 are external nets. Note that, it is always possible to derive an equivalent representation of Π as an *m-way* **net** partition $\Pi = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\} = \{\mathcal{N}_1, \ldots, \mathcal{N}_m; \mathcal{N}_{ext}\}$, the set $\mathcal{N}_h$ for $h = 1, \ldots, m$ contains internal nets connecting part $\mathcal{U}_h$, whilst $\mathcal{N}_{ext}$ all the external nets. The number of cut nets corresponds to the number of nets in $\mathcal{N}_{ext}$, and denotes the *cut-size* of the partition. The net partition $\Pi = \{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3; \mathcal{N}_{ext}\}$ corresponding to the node partition in Figure 4.2b is $\mathcal{N}_1 = \{n_1\}$, $\mathcal{N}_2 = \{n_5\}$, $\mathcal{N}_3 = \{n_4\}$, and $\mathcal{N}_{ext} = \{n_2, n_3\}$. The cut-size of Π is, hence, equal to 2.

A partition $\Pi = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$ is called ε-balanced if each part $\mathcal{U}_h$ satisfies the balance constraint:

$$|\mathcal{U}_h| \leq (1 + \varepsilon) \left\lceil \frac{\sum_{h=1}^{m} |\mathcal{U}_h|}{m} \right\rceil, \tag{4.2}$$

where $|\mathcal{U}|$ denotes the cardinality of set $\mathcal{U}$. When $\varepsilon = 0$, the partition is said to be *balanced*. An optimal, ε-balanced m-way partition of hyper-graph $\mathcal{H}$ is a partition satisfying (4.2) and with minimum *cut-size*.

Given a constraint matrix $\tilde{A}$, two equivalent hyper-graph representations can be derived: a row-net or a column-net model. Without loss of generality, we describe only the row-net representation of $\tilde{A}$, since a column-net model of $\tilde{A}$ can be obtained as a row-net model of its transpose $\tilde{A}^\top$.

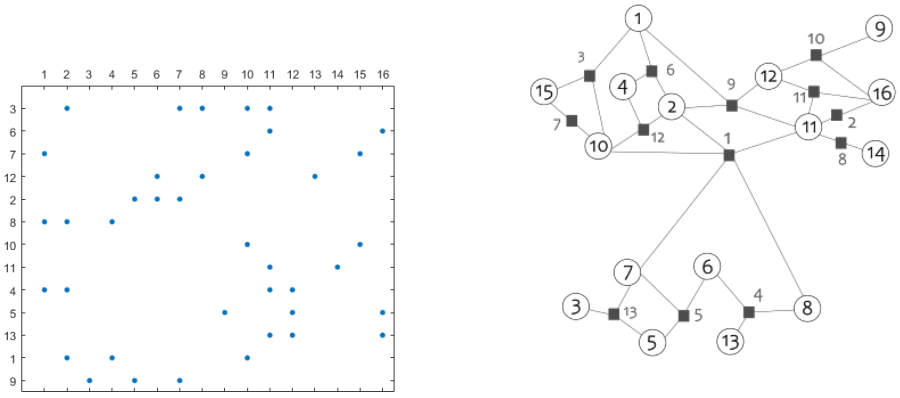
A *hyper-graph row-net model* of matrix $\tilde{A}$ is a hyper-graph $\mathcal{H}_{\tilde{A}} = (\mathcal{U}, \mathcal{N})$ where each node in $\mathcal{U}$ corresponds to a column of $\tilde{A}$ and each net in $\mathcal{N}$ corresponds to a row of $\tilde{A}$. More specifically, if the entry a_{ij} of $\tilde{A}$ is non-zero, then node u_j is a pin of net n_i. Since each row of $\tilde{A}$ corresponds to a constraint and each column to a decision variable of our MILP, then the nodes of the hyper-graph represent the decision variables and a net represents a constraint (i.e. a connection) between decision variables. Note that permuting rows and columns of $\tilde{A}$ does not change the hyper-graph structure but only the labels associated to nodes and nets. An *m-way* partition $\Pi_{\tilde{A}} = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\} = \{\mathcal{N}_1, \ldots, \mathcal{N}_m; \mathcal{N}_{ext}\}$ of the row-net hyper-graph representation of matrix $\tilde{A}$ induces a permutation of $\tilde{A}$ and the permuted matrix $\tilde{A}^\Pi$ 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 for $h = 1, ..., m$ in (4.1), whilst nets in $\mathcal{N}_{ext}$ correspond to the border $[A_1, \ldots, A_m]$. Formal derivation of the result is provided in [29].

In Figure 4.3b we show the hyper-graph now-net model of the (sparse) constraint matrix represented in Figure 4.3a. Column labels refers to nodes and row labels refers to nets. In Figure 4.4b we report a possible 3-way partition of the hyper-graph in Figure 4.3b and the resulting (permuted) constraint matrix in Figure 4.4a.

4.3 Proposed partitioning algorithm

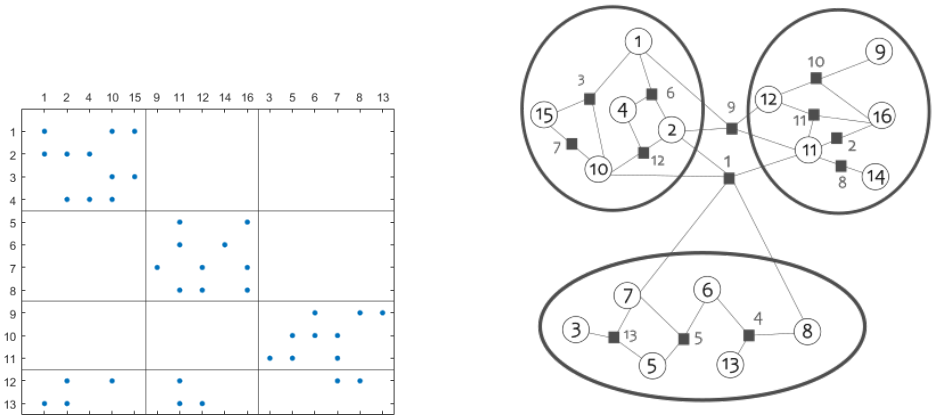
4.3.1 Procedure outline

State-of-the-art hyper-graph partitioning algorithms cannot be applied directly to our setting because they do not account for the following key



(a) **(b)**

Figure 4.3: Constraint matrix $\tilde{A}$ (a) and its row-net hyper-graph representation $\mathcal{H}_{\tilde{A}}$ (b).



(a) **(b)**

Figure 4.4: 3-way partition of the $\mathcal{H}_{\tilde{A}}$ (b) and induced singly-bordered block-angular form $\tilde{A}^{\Pi}$ of matrix $\tilde{A}$ (a).

requirements of

1. maximizing the number m of agents while minimizing the size p of the border matrix representing the coupling constraints and

2. evenly distribute the number of integer decision variables among the agents.

More specifically, the performance guarantees provided by the decentralized solution summarized in Algorithm 1 are affected by the *border-to-agents* ratio $\frac{p}{m}$. In principle, the smaller the border, compared with the number of agents, the better the performance of the obtained approximate solution. Minimization of the border size p is already promoted by standard partitioning algorithms. However, the number of parts m is typically considered as fixed and a-priori given. In our context, it is not known and its estimation (and maximization) becomes crucial and must be integrated in the partitioning algorithm.

The second requirement, instead, ensures even distribution of the computational load in the m lower-dimensional MILPs associated with the agents. The standard notion of balanced partition must then be extended to account for the mixed-integer nature of the set of nodes in the hyper-graph $\mathcal{H}_{\tilde{A}}$. Since each node is associated with a decision variable that can be either continuous or discrete, a balanced m-way partition must be such that discrete nodes are distributed in equal number among the agents, whereas continuous nodes can be unevenly distributed if this helps in balancing the discrete ones and/or in reducing the size of the border.

Building upon available algorithms, we propose a novel technique to bring a matrix $\tilde{A}$ into a singly-bordered block-angular form via hyper-graph partitioning, whilst accounting for the above mentioned requirements. Algorithm 2 provides the pseudo-code of the partitioning procedure for a fixed number of parts m. Some strategies for its identification are proposed and described in Section 4.3.6.

The algorithm takes as input the hyper-graph $\mathcal{H} = (\mathcal{U}_c \cup \mathcal{U}_d, \mathcal{N})$ to be

Algorithm 2 Hyper-graph partitioning

Input:

$\mathcal{H}$	original hyper-graph	$D_{\max}$	max discrete nodes
$\mathcal{U}_d$	Set of discrete nodes	m	number of parts
$\mathcal{U}_c$	Set of continuous nodes	π	threshold for the $\frac{p}{m}$ ratio
a	penalty parameter		

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: $\Pi^\star = \emptyset$
4: $p = 0$
5: **repeat**
6: $n_\delta = h \cdot D_{\max} - |\mathcal{U}_d|$
7: $\mathcal{H}_\rho =$ `add_dummy_nodes`$(\mathcal{H}_\rho, n_\delta)$
8: $[\mathcal{M}, \widetilde{\mathcal{H}}_\rho] =$ `assign_continuous2discrete`$(\mathcal{H}_\rho, \mathcal{U}_d, \mathcal{U}_c)$
9: $\widetilde{\Pi}_\rho = \{\mathcal{U}_i\}_{i=1}^m = \{\{\mathcal{N}_i\}_{i=1}^m; \mathcal{N}_{ext}\} =$ `h_way_partitioning`$(\widetilde{\mathcal{H}}_\rho, h, D_{\max}, a)$
10: $[\mathcal{H}_\rho, \Pi_\rho] =$ `ungroup_disc_continuous`$(\widetilde{\mathcal{H}}_\rho, \widetilde{\Pi}_\rho, \mathcal{M})$
11: $[\beta_1, \ldots, \beta_B, \mathcal{N}_\tau] =$ `isolateBlocks`$(\mathcal{H}_\rho, \Pi_\rho)$
12: $[\mathcal{H}_\rho, \Pi_\rho] =$ `remove_dummy_nodes`$(\mathcal{H}_\rho, \Pi_\rho, n_\delta)$
13: $p = p + |\mathcal{N}_\tau|$
14: $\Pi^\star = \Pi^\star \cup \{U_{\beta_1}, \ldots, \mathcal{U}_{\beta_B}\}$
15: $h = h - B$
16: $\mathcal{H}_\rho = \left(\mathcal{U}_\rho \setminus \left\{\bigcup_{b=1}^B \mathcal{U}_{\beta_b}\right\}, \mathcal{N}_\rho \setminus \left\{\bigcup_{k=1}^B \mathcal{N}_{\beta_k}\right\} \setminus \mathcal{N}_\tau\right)$
17: **until** $h < 2 \vee \frac{p}{m} > \pi$

Output: Partition $\Pi^\star$

partitioned, with information on whether a node represents a continuous (inside $\mathcal{U}_c$), or discrete (inside $\mathcal{U}_d$) decision variable, and the number m of desired parts. Three additional parameters are supplied:

- the maximum number $D_{\max}$ of discrete variables admitted per block, which measures the maximum computational effort that can be endured by each agent;
- a threshold π denoting a satisfactory border-to-agents ratio, representing the desired performance guarantees provided by Algorithm 1
- a penalty parameter $a \leq 1$, which enters in the definition of the objective function used to guide the partitioning algorithm.

The idea behind the overall procedure is as follows. The algorithm seeks for an m-way partition $\Pi^\star$ of $\mathcal{H}$ by iteratively isolating groups of nodes (the parts). Parts that are identified in one iteration are removed and a reduced hyper-graph $\mathcal{H}_\rho$ is passed on to the next iteration to seek for an h-way partition with $h < m$. Each iteration unfolds as follows. At the beginning, the algorithm performs a preliminary clustering operation during which each continuous variable is assigned to the discrete variable it is most coupled with (cf. Step 7). In the resulting coarsened hyper-graph $\widetilde{\mathcal{H}}_\rho$, each node (also called *supernode*) contains one discrete variable only and the associated continuous ones. Then, the algorithm performs an h-way ε-balanced minimum cut-size partition $\widetilde{\Pi}_\rho = \{\mathcal{U}_1, \ldots, \mathcal{U}_h\}$ of the coarsened hyper-graph $\widetilde{\mathcal{H}}_\rho$ (cf. Step 8) and ungroups the supernodes to express the obtained partition in terms of a partition Π_ρ of the hyper-graph $\mathcal{H}_\rho$ (cf. Step 9). The amount of imbalance ε in the partitioning phase is ensured by adding an appropriate amount of fictitious nodes (cf. Step 6) at the beginning, which are then removed at the end of the iteration (cf. Step 11).

The one-to-one correspondence between discrete nodes in $\mathcal{H}_\rho$ and supernodes in $\widetilde{\mathcal{H}}_\rho$ allows to evenly distribute (within the ε tolerance) the discrete variables and has the added bonus of simplifying the partitioning

phase by reducing the dimension of the hyper-graph, as nodes associated to continuous variables are not present in $\widetilde{\mathcal{H}}_\rho$. Clearly, nets of $\widetilde{\mathcal{H}}_\rho$ are different from the nets in $\mathcal{H}_\rho$ as they must account also for indirect connections between discrete nodes that are not directly connected in $\mathcal{H}_\rho$, but they are connected to the same continuous variable. After the ungrouping phase, the B-least coupled parts $\mathcal{U}_{\beta_1}, \ldots, \mathcal{U}_{\beta_B}$ in Π_ρ are isolated (cf. Step 10) and all nets connecting isolated parts are removed. The number of external nets p is updated (cf. Step 12), parts $\mathcal{U}_{\beta_1}, \ldots, \mathcal{U}_{\beta_B}$ are stored and will be parts of the returned partition $\Pi^\star$ (cf. Step 14). Their nodes and associated nets are removed from $\mathcal{H}_\rho$ (cf. Step 15), and the algorithm proceeds to the next iteration.

Such a coarsening-partitioning-isolation routine is performed iteratively until all m parts are identified. Since the size of the border p cannot decrease across the iterations, any execution leading to an unsatisfactory final partition can be preemptively interrupted monitoring the border-to agents ratio $\frac{p}{m}$. *Early-stop* occurs whenever $\frac{p}{m}$ exceeds a quality-threshold π, that can be tuned based on qualitative knowledge of the application and/or explicit specifications, when available.

Notice that, the combination of the m-way partitioning scheme together with the block-isolation procedure makes the approach powerful since it allows to improve the final partition whilst reducing the dimension of the hyper-graph as iterations progress, and thus the run-time required by each execution.

A more detailed description of the operations performed at each iteration is provided in the following Sections 4.3.2-4.3.5.

4.3.2 Initialization and introduction of fictitious nodes

To allow for an unbalanced partition, we follow the approach suggested in [31], which consists in adding a specific number of dummy nodes (i.e., nodes not connected by any net) and then seek for a balanced partition.

Without any prior knowledge on the hidden multi-agent structure of our

MILP, we can tune the imbalance based on our computational constraints. To this end, consider the maximum amount $D_{\max}$ of discrete variables that are admitted per block and the tentative value m for the size of the partition. In principle, we can accommodate a total of $mD_{\max}$ discrete decision variables equally distributed among m parts. But if the number n_d of discrete decision variables in our MILP is less than $mD_{\max}$, then we can add

$$n_\delta = mD_{\max} - n_d \tag{4.3}$$

dummy discrete decision variables and then perform a balanced m-way partition to provide some flexibility. Indeed, those agents that will receive some dummy discrete variables will have a lower computational load, but none of them will receive more than $D_{\max}$ discrete variables.

Knowing that an m-way balanced partitioning algorithm run on a graph with $mD_{\max}$ will produce a partition with $|\mathcal{U}_i| = D_{\max}$ we can use (4.2) to compute the relative imbalance

$$\varepsilon = \frac{D_{\max}}{\left\lceil \frac{n_d}{m} \right\rceil} - 1 \tag{4.4}$$

between the parts that this procedure allows.

Alternatively, if an estimate $\hat{\varepsilon}$ for ε is known, it can be used to add $n_\delta = n_d\hat{\varepsilon}$ variables provided that $n_d + n_\delta \leq D_{\max}$.

4.3.3 Continuous-discrete nodes association

Merging continuous and discrete nodes before computing a partition of the hyper-graph is fundamental in order to focus on distributing the discrete variables among the parts. Indeed, the grouping operation allows to create a *coarsened* hyper-graph in which each supernode represents one discrete decision variable and all its associated continuous variables. Clearly, the nets connecting the nodes in this coarsened hyper-graph account also for constraints between continuous variables associated to different discrete ones, whilst constraints involving only continuous variables and the discrete variable to which they are associated disappear.

Algorithm 3 assign_continuous2discrete

Input:				
	$\mathcal{H}$	original hyper-graph	$\mathcal{U}_d$	set of discrete nodes
	$\Gamma : \mathcal{U} \to 2^{\mathcal{U}}$	neighbours map	$\mathcal{U}_c$	set of continuous nodes

$\mathcal{M} = \{\}$
$w = \mathbf{1} \in \mathbb{R}^n$
$\widetilde{\mathcal{H}} = \mathcal{H} = (\mathcal{U}_c, \mathcal{U}_d)$
for each $v \in \mathcal{U}_c$ **do**
  if $\Gamma(v) \cap \mathcal{U}_d \neq \emptyset$ **then**
    // cycle through the discrete neighbours of v
    for each $u \in \Gamma(v) \cap \mathcal{U}_d$ **do**
      compute $r(u, v)$ according to (4.7)
    end for
    select $u^\star = \underset{u \in \Gamma(v) \cap \mathcal{U}_d}{\arg\max} r(u, v)$
  else if $\Gamma(v) \cap \mathcal{U}_c \neq \emptyset$ **then**
    // cycle through the continuous neighbours of v
    for each $u \in \Gamma(v) \cap \mathcal{U}_c$ **do**
      compute $r(u, v)$ according to (4.7)
    end for
    // find the best contraction partner
    select $u^\star = \underset{u \in \Gamma(v) \cap \mathcal{U}_c}{\arg\max} r(u, v)$
  else
    randomly select $u^\star \in \mathcal{U}_d$
  end if
  $[\mathcal{M}, \widetilde{\mathcal{H}}, \Gamma, w] = \texttt{contract}(\widetilde{\mathcal{H}}, \Gamma, w, u^\star, v)$
end for

Output:				
	$\mathcal{M}$	contraction map	$\widetilde{\mathcal{H}}$	coarsened hyper-graph
	Γ	updated neighbours map	w	updated array of weights

The aggregation of nodes generates a lower-dimensional hyper-graph, which simplifies the effort involved in partitioning the graph. However, it also changes the structure of the connections between the discrete nodes and a not appropriate assignment may hamper the partitioning phase. For this reason, the adopted *grouping criterion* must be carefully selected in order to minimize the impact on the structure of the connections and ensure that a good partition of the resulting coarsened hyper-graph yields to a good partition of the original one.

The grouping policy proposed is summarized in Algorithm 3, which is a modified version of an existing *coarsening* procedure, known as *Edge Coarsening (EC)* and adopted in [32, 33, 34] to reduce the dimensions of the input hyper-graph and improve the performance of the partitioning algorithm.

EC merges pairs 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). In particular, it assigns to each net n in the hyper-graph a *hyper-edge weight* $\omega(n)$, measuring the strength of the connection established between any pair of nodes $u, v \in \text{Pins}(n)$ by net n and computed as:

$$\omega(n) = \frac{1}{|\text{Pins}(n)|}. \tag{4.5}$$

The smaller the size of the net n, the higher its hyper-edge weight $\omega(n)$ and the tighter the coupling between its pins. The strength of the connection between a pair of nodes u, v can then be measured using the *rating function*

$$r_{\text{EC}}(u, v) = \sum_{n \in \text{Nets}(u) \cup \text{Nets}(v)} \omega(n) \tag{4.6}$$

which can be used to determine, for each node u, the best *contraction partner* $v^\star = \arg\max_v \{r(u, v)\}$ to be merged with. The grouping operation is, then, performed as follows: EC randomly selects a node u in the hyper-graph, it searches its best contraction partner $v^\star$ among its neighbours and merge them together. The procedure is repeated until a given dimension/density criterion on the coarsened hyper-graph is met.

The grouping strategy in Algorithm 3 follows a similar paradigm, but adopts restrictions to avoid merging together discrete nodes. Specifically, it randomly selects a continuous node v in the hyper-graph in Step 4 and looks for a contraction partner u among its discrete neighbours (cf. Steps 5-8). If no discrete neighbour exists, a continuous contraction partner is selected (cf. Steps 10-13). In case of completely isolated node, the discrete counterpart is chosen arbitrarily (cf. Step 16). Candidate contraction partners u, either discrete or continuous, are evaluated and *ranked* in Steps 7 and 12, according the a modified version $r(u, v)$ of the rating function in (4.6):

$$r(u,v) = \frac{1}{w(u) + w(v)} \sum_{n \in \text{Nets}(u) \cup \text{Nets}(v)} \omega(n) \tag{4.7}$$

where the function $w(u)$ defines the *weight* of the supernode u and is equal to the number of nodes inside the supernode, with $w(u) = 1$ when u is a node and not a supernode. The rating function $r(u, v)$ is inspired to the one used in EC but contains an additional scaling factor $\frac{1}{w(u)+w(v)}$ to promote a balanced distribution of the continuous variables in the super-nodes. Once the best candidate $u^\star$ is found (see Steps 9 and 14), nodes $u^\star$ and v are grouped together, the structure of the hyper-graph is updated and information for the ungrouping operation is stored in a table $\mathcal{M}$. The procedure is repeated until all continuous nodes are assigned.

4.3.4 h-way partitioning

At the core of iterative procedure, the partitioning algorithm distributes the discrete variables (along with their associated continuous one obtained from the previous phase) among the agents following a *sequential break-off* paradigm. The procedure is summarized in Algorithm 4.

First introduced in [35], 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. In particular, at each iteration k of the sequential

Algorithm 4 h-way partitioning (via sequential break-off)

	$\widetilde{\mathcal{H}}$	coarsened hyper-graph		
Input:	h	number of parts	a	penalty parameter
	$D_{\max}$	max discrete nodes	n_d	number of discrete nodes

$n_h = \left\lceil \frac{n_d}{h} \right\rceil$
$\mathcal{U}_2 = \mathcal{U}$
for $k = 1, \ldots, h-1$ **do**
  $\mathcal{U}_1 = \texttt{select\_n\_h\_elements}(\mathcal{U}_2, n_h)$
  $\mathcal{U}_2 = \mathcal{U}_2 \setminus \mathcal{U}_1$
  $\Pi_{\text{rnd}} = (\mathcal{U}_1, U_2)$
  $(\mathcal{U}_1^\star, \mathcal{U}_2^\star) = \texttt{FM\_w\_pen}(\widetilde{\mathcal{H}}, \Pi_{\text{rnd}}, D_{\max}, 1)$
  $\mathcal{U}_{\Pi_0,j} = \mathcal{U}_1^\star$
  $\mathcal{U}_2 = \mathcal{U}_2^\star$
end for
$\Pi_0 = (\mathcal{U}_{\Pi_0,1}, \ldots, \mathcal{U}_{\Pi_0,h})$
$\Pi^\star = \texttt{pairwise\_comparison}(\Pi_0, A_\rho, D_{\max})$

Output: $\Pi^\star$ Final Partition

Algorithm 5 Iterative Refinement Bisection Algorithm

	$\widetilde{\mathcal{H}}$	reduced hyper-graph		
Input:	Π_0	Initial partition	a	scaling factor
	$D_{\max}$	max discrete nodes	n_d	number of discrete nodes

1: Δ_{cs} = `compute_cut_size` (Π_0)
2: $\Pi_{pass} = \Pi_0$
3: **repeat** // pass
4: i=1
5: $\Pi_{list}[i] = \Pi_{pass}$
6: **repeat** // move
7: **for each** $u \in \widetilde{\mathcal{U}}$ **do**
8: $\Delta_g(u)$ = `compute_pen`$(\widetilde{\mathcal{H}}, D_{\max}, a, \Pi_{list}[i])$
9: $g(u)$ = `compute_gain`$(\Pi_{list}[i], a, D_{\max})$ - Δ_g
10: **end for**
11: $[mv^\star]$ = `best_feasible_move`$(g, \Delta_g, \Pi_{list}[i], D_{\max})$
12: $\Pi_{list}[i+1]$ = `move`$(mv^\star, \Pi_{list}[i])$
13: $i = i + 1$
14: `lock_moved_node`$(mv^\star)$
15: **until** all nodes are locked // end move
16: $\Pi^\star$ = `find_best_partition`(Π_{list})
17: Δ_{cs} = `compute_cut_size`$(\Pi_{list}[1])$-`compute_cut_size`$(\Pi^\star)$
18: **if** $\Delta_{cs} \geq 0$ **then**
19: $\Pi_{pass} = \Pi^\star$
20: **end if**
21: **until** $\Delta_{cs} \geq 0$ // end pass

Output: Π_{pass} Final Bisection

break-off routine an initial 2-way partition (bisection) $\{\mathcal{U}_1, \mathcal{U}_2\}$ is created (cf. Steps 4-5) by randomly selecting a subset $\mathcal{U}_1$ of $n_h = \left\lceil \frac{n}{h} \right\rceil$ nodes. This choice establishes a relative unbalance between the parts equal to:

$$\varepsilon_2 = \frac{|\mathcal{U}_2|}{\left\lceil \frac{|\mathcal{U}_1|+|\mathcal{U}_2|}{2} \right\rceil} - 1 \tag{4.8}$$

Such bisection is, then, optimized (cf. Step 7) to reduce the cut-size preserving the ε_2 relative unbalance, the refined subset of $\mathcal{U}_1^\star$ is set aside and the procedure is repeated on the remaining elements in $\mathcal{U}_2^\star$ until all h parts of the h-way partition are stored. Finally, the procedure performs pair-wise comparisons between the parts (cf. Step 12) aiming at making the h-way partition *pair-wise optimal*, i.e. such that for every pair of parts $\mathcal{U}_i, \mathcal{U}_j$ for $i = 1 < j \leq h$ the 2-way partition $\{\mathcal{U}_i, \mathcal{U}_j\}$ is a 0-balanced minimum cut-size bisection of the sub-hyper-graph induced by $\mathcal{U}_i$ and $\mathcal{U}_j$. Comparisons are performed iteratively until either a maximum number of iterations is reached, or all the possible $\binom{h}{2}$ pairs of subsets $\mathcal{U}_i, \mathcal{U}_j$ are compared but no move is found to improve the partition.

A minimum cut-size bisection can be obtained starting from the initial random bisection, by applying a modified version an *iterative refinement* partitioning scheme proposed by Fiduccia and Mattheyses (FM) in [35], summarized in Algorithm 5. As in the original version, the algorithm starts from an initial partition Π_0 of the hyper-graph and iteratively refines it moving nodes between the parts to reduce the hyper-edge cut. The procedure is organised in incrementally improving linear-time *passes*. During each pass (cf. Steps 4-20), every node is moved exactly once and nodes are processed according to a minimum cut-size oriented order. At the end of the pass the sub-sequence providing the best improvement is performed, the new partition is stored (cf. Steps 16 and 19) and a new pass is started. No further pass is performed if no improvement is obtained (i.e. $\Delta_{cs} \leq 0$).

The order in which nodes are moved from one part to the other is decided according to a *gain* g associated to the move. A pass consists, then,

in iteratively selecting the feasible move (i.e. not compromising the balance requirements) with the highest gain, perform it, *lock* the involved node (so that it/they cannot be moved again in the same pass) and update the gains of the remaining moves. Note that negative gains do not prevent moves to be executed, thus making the algorithm able to perform hill-climbing and escape locally optimal partitions.

In a standard FM scheme, the gain of a move coincides with the reduction of the cut-size obtained by moving the corresponding node to the complementary part of the current partition. In the proposed bisection algorithm we add a penalisation Δ_g to the cut-size reduction metric (cf. Steps 8-9), in order to discourage moves that improve the current partition by favouring the aggregation of real discrete variables in a single part, while they should be associated to different agents. This possibility typically arises when $D_{\max}$ is large enough and the number of fictitious variables added to the hyper-graph provide an excessively high flexibility that allows the procedure to group discrete variables in the same part in order to reduce the cut-size, with the undesired effect of increasing the computational load of the resulting aggregated agent.

In particular, denote with $\Theta : 2^{\mathcal{U}} \to \mathbb{N}$ the map returning the number of real (non-dummy) discrete nodes in any subset of nodes $\mathcal{U}$ and let $\mathcal{U}_1$, $\mathcal{U}_2$ be the two sides of the current bisection to be refined by the algorithm. If $u \in \mathcal{U}_1$, then the penalisation Δ_g is given by

$$\Delta_g(u) = a \cdot \frac{\left| \frac{\Theta(\mathcal{U}_1)}{h-i} - \left\lceil \frac{n_d}{m} \right\rceil - \Theta(\{u\}) \right|}{D_{\max}} \tag{4.9}$$

where h takes its value from the iteration of Algorithm 2, i denotes the current iteration of the sequential break-off procedure in Algorithm 5, and $a \leq 1$ is a scaling factor that can be tuned to reduce the order of magnitude of the penalisation. The choice $a = 0$ recovers the standard FM algorithm, whilst $a = 1$ provides the maximum impact.

The rationale behind this penalization is to avoid clustering all discrete variables. Indeed if the average number of real discrete decision variables

($\Theta(\mathcal{U}_1)$ normalized by the expected number of agents $h-i$ inside $\mathcal{U}_1$) in $\mathcal{U}_1$ is above the expected average $\left\lceil \frac{n_d}{m} \right\rceil$, then $\Delta_g(u)$ will promote the removal of real nodes from $\mathcal{U}_1$. Otherwise, it will discourage the removal of a real variable from $\mathcal{U}_1$. The discussion is symmetric in case $u \in \mathcal{U}_2$.

Normalisation of (4.9) through $D_{\max}$ ensures that $\Delta_g(u)$ is effective only when all moves have no effect on the number of coupling constraints.

Once the aggregated nodes are distributed between the h agents, the corresponding h-way partition of the original hyper-graph is retrieved by means of a un-grouping operation exploiting the contraction information in $\mathcal{M}$. Notice that the final partition is not guaranteed to have minimum cut-size, because Algorithm 4 is based on a heuristic strategy. Indeed, solving the h-way hyper-graph partitioning problem to optimality would be NP-hard [36], thus requiring enumerative resolution schemes, impractical in almost any application.

4.3.5 Block isolation

The partition obtained via Algorithm 4 is generally sub-optimal and highly dependent on the size of the hyper-graph, the way in which continuous and discrete nodes are merged and the starting partition, which is randomly chosen. Thus, multiple runs of the procedure with different initializations are likely to produce different results.

In order to exploit this inherent randomization to improve the quality of the final partition, we do not return the partition provided by Step 8 of Algorithm 2, but we isolate the Q-least connected parts and re-run the procedure on the remaining portion of the hyper-graph. The block isolation procedure works as follows.

We first determine the number of cut nets connecting each part $\mathcal{U}_i$, $i = 1, \ldots, h$. We then select the part $\mathcal{U}_{i^\star}$ having the minimum number of nets in the cut. $\mathcal{U}_{i^\star}$ is then added to the output partition $\Pi^\star$ and removed from $\mathcal{H}_\rho$ together with its internal and external nets. Any other block that after this removal operation is no longer connected by any external net

is isolated and also added to the final partition. In this case, nodes and internal nets of such isolated blocks are also removed from $\mathcal{H}_\rho$.

4.3.6 Estimation of the number of blocks

Algorithm 2 partition the input hyper-graph $\mathcal{H}_{\widetilde{A}}$ in a number of blocks 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 two strategies for such estimation, that fit the considered setting and objectives.

The first option consists in running Algorithm 2 multiple times with different tentative m and selecting the best estimate $\hat{m}$ as the value providing the partition with the smallest border-to-agents ratio.

Without a careful choice of the candidates, blindly testing all values m in a given range is ineffective and time-consuming. In fact, even with adequate restrictions of the interval of admissible m, obtaining a correct estimate may require many executions. That is why the proposed strategy accelerates the m-identification by exploiting the additional degrees of freedom due to the presence the fictitious variables to avoid performing unnecessary tests.

More in detail, the maximum number of dummy nodes $n_\delta|_{\max}$ that can be introduced in the hyper-graph grows with the number of parts m. As a consequence, for tentative values m lower than the correct estimate m_0 the algorithm has little flexibility and is hence likely to attribute at least one real variable to each subset of the final partition, eventually using the available dummies to allow for an unbalanced distribution. However, as the number of parts increases more fictitious nodes are introduced in the hyper-graph making the procedure able to entirely fill surplus parts with unnecessary dummy variables whenever m exceeds m_0.

The overall estimation can be, thus, carried out by spanning a range $\left[\left\lfloor \frac{n_d}{D_{\max}} \right\rfloor, \left\lceil \frac{n_d}{D_{\min}} \right\rceil\right]$ of admissible values for m, estimated from the expected maximum and minimum number of discrete variables per agent, $D_{\max}$ and

$D_{\min}$. Candidate m can be selected according to either a *top-down* or a *bottom-up* policy.

A *top-down* approach starts testing the highest value $m = \left\lceil \frac{n_d}{D_{\min}} \right\rceil$ and iteratively reduces m according to the number of parts in the obtained partition that are containing dummy variables only. Iterations stops whenever m cannot be further reduced without increasing the size of the border p. A *bottom-up* policy, instead, takes as first tentative $m = \left\lfloor \frac{n_d}{D_{\max}} \right\rfloor$ and increases it with a fixed step until the partition obtained contains at least one surplus group of dummy variables only. In both cases, the procedure returns the partition of the input hyper-graph $\mathcal{H}_{\tilde{A}}$ with the smallest border-to-agents ratio, net of the dummy variables, and the corresponding estimated number of agents $\hat{m}$.

The choice between the two *exploration criteria* should be application-driven. In general, a top-down approach produces the final estimate with fewer tests at the price of an increased partitioning time per value m, whilst a bottom-up one may require more iterations but with reduced run-time per tentative. This is due to the introduction of dummy nodes that make the size of the hyper-graph increase, as well as the time required by the sequential break-off partitioning operation to produce a final result. In addition, the computational time per tentative is also affected by the number of parts, being the worst-case maximum number of iterations of the coarsening-partitioning-isolation algorithm linearly increasing with with m.

A correct estimation of parameters $D_{\max}$ and $D_{\min}$ is, instead, crucial to limit the growth of the hyper-graph. Indeed, when all $n_\delta|_{\max}$ dummy variables are introduced, the higher $D_{\max}$, the higher the expansion rate for increasing m. $D_{\max}$ should be, thus, relatively small to avoid an unnecessary increase of the overall computational time, but large enough to provide the flexibility needed to retrieve a satisfactory partition. Notice that $D_{\min}$ does not have a direct impact on the quality of the result, but only contributes to set the upper-bound for candidate values m.

The second approach we proposed for the identification of the number of agents m builds a *tree* of partitions by repeatedly applying Algorithm 2 on the input hyper-graph $\mathcal{H}_{\widetilde{A}}$ and its sub-graphs.

The procedure starts computing K k-way partitions of the original hyper-graph, considering as candidate k the first K prime numbers $\mathcal{C}_K$. Each tentative is associated to a node ν of the tree and produces a partition Π_ν of k parts with cut-size p_ν. The operation is, then, recursively repeated at each node where, all k-way partitions, with $k \in \mathcal{C}_K$, are computed for each sub-hyper-graph $\mathcal{H}_{\nu,i}$ *induced* by part i of Π_ν (i.e. the hyper-graph obtained considering only nodes and internal nets of part i of Π_ν). The recursion stops whenever the cumulative size of the border p becomes excessively high compared with total the number of agents. Thus, a condition on the border-to-agents ratio $\frac{p}{m} < \pi$ determines whether or not *closing* a branch and stop partitioning.

Notice that, whilst information on p can be easily propagated from a node to its *children*, determining the total number of parts to be used in the evaluation of $\frac{p}{m}$ is not straightforward since the final m cannot be known a priori, and must be then efficiently estimated. We here adopt a trivial solution that considers a *target* number of parts m_{target} equal for all the nodes. More sophisticated approaches are mentioned in Section 6.2.

Exploration of the resulting *k-ary* tree can be, thus, optimized by pruning a branch whenever the estimated border-to-agents ratio does not meet prescribed requirements, specified by the threshold π, or is not better than the best result achieved.

The progressive reduction of the size of the hyper-graph at each level of the recursion yields better performance of the partitioning algorithm. This motivates the *depth-first* policy adopted to traverse the tree, that gives priority to the creation of children-nodes with respect to iteration of k. By doing so, satisfactory lower-bounds are likely to be found sooner in the search, increasing the pruning opportunities and reducing the computational effort required by the overall m-identification process.

A good trade-off between computational time and overall quality of the partition calls for a fine tuning of the threshold π, to be made considering the decomposition phase as well as the performance of the decentralized optimization algorithm. Partitions with a border-to-agents ratio above the threshold are assumed to yield a poor-quality solution of the MILP problem and, thus, discarded.

4.4 Alternative approaches

The procedure proposed in Algorithm 2 combines features of different state-of-the-art partitioning schemes. It does not, however, fit into the classification suggested in [32], that roughly identifies three types of algorithms:

- Iterative Refinement Partitioning Algorithms, as the already mentioned FM [35] or the Schweikert - Kernighan - Lin (SKL) [37], that start from an initial partition of the hyper-graph and repeatedly refine it moving nodes across the parts to reduce the cut-size.

- Coarsening-Partitioning Algorithms [38, 39] that perform a preliminary clustering operation to group highly-connected nodes and retrieve the partition of the hyper-graph by means of iterative refinement schemes.

- Multilevel Algorithms as the ones in [40] and [32], that perform coarsening and un-coarsening operations before and after the partitioning stage, according to a multi-level paradigm, to exploit randomisation while refining the partition.

In fact, a single iteration of Algorithm 2 applies a strategy similar to a coarsening-partitioning technique, but with a different rationale behind the clustering operation. In particular, standard approaches merge nodes with the main objective of reducing the size of the hyper-graph and

improve efficiency in the partitioning phase. They can either merge pairs of highly connected nodes, following an *edge-coarsening* policy, or adopt an alternative *hyper-edge-coarsening* one that, instead, groups subsets of pins of a net. In the proposed scheme, however, the clustering step is leveraged to uniformly distribute discrete nodes between the parts and not to reduce the hyper-graph size.

Another distinctive feature of the proposed procedure is the sequential break-off paradigm employed to distribute the nodes of the coarsened hyper-graph. Indeed, state-of-the-art multi-level partitioning algorithms typically retrieve the sought m-way partition by first computing a bisection of the initial graph, and then recursively repeating the operation on the two sub-hyper-graphs obtained. Although appealing, following a similar rationale in the considered set up makes it difficult to simultaneously account for the limitations on the maximum number of discrete variables admitted per agent and allow for unbalanced distributions. An adaptive tolerance parameter ε, as the one used in [34] could potentially solve the problem, however finding a law to adequately describe its evolution across each recursion level may be not trivial and requires further investigation.

Lastly, the integration of the coarsening-partitioning phase with the block-isolation procedure allows to reduce the cut-size of the obtained partition across the iterations while ensuring a progressive decrease in the size of the hyper-graph. This defines a novel competitive refinement strategy that proves to be more effective and better suited for the set-up considered in this thesis than other existing schemes. In particular, it compares with the two well-established *iterative* [35] and *hyper-edge refinement* [32] approaches that, instead, try to improve a partition by moving respectively single nodes or entire subsets of pins of a net, in the attempt of reducing the number of hyper-edges in the cut. Such schemes are typically very efficient when applied to refine bisections but have poor performances on generic m-way partitions. This is due to the fact that the iterative refinement typically have difficulties in reconstructing nets

having pins in more than one part, whilst hyper-edge refinement - that overcomes such limitation - is not suited to enforce balance requirements. The same holds for the *multi-level* refinement strategies, *V-Cycle*, *v-Cycle*, or *vV-Cycle*, proposed in [32] that combine the basic iterative refinements with additional coarsening-uncoarsening steps. Therefore, conventional refinement techniques may yield little-to-no improvement when used to refine the partition obtained via sequential break-off, as that returned by Step 10 of Algorithm 2. Block isolation is, instead, generally more effective since it simplifies the hyper-graph at every new iteration while exploiting the randomization of the clustering phase to generate different coarsened hyper-graphs and, thus, possibly different partitions.

Chapter 5

Performance assessment of the proposed resolution scheme

5.1 Introduction

In this chapter, we shall assess the effectiveness of the strategy proposed in Chapter 4 for recovering the multi-agent structure of a MILP, by considering randomly generated constraint matrices $\tilde{A}$ of a MILP (2.1) with a *hidden* (i.e., not evident) singly-bordered block-diagonal structure. A random matrix with such a structure can be easily generated by starting from a singly-bordered block-diagonal one (with prescribed number of blocks and number of coupling constraints) and randomly permuting its rows and columns.
In the sequel, different configurations are tested with a twofold purpose of

- investigating the impact of the input parameters $D_{\max}$ (maximum amount of discrete variables that are admitted per block) and m (number of blocks) on the procedure, both in terms of quality of the obtained result and run time;

- providing some insight on how to tune the two parameters.

The quality of the obtained results is evaluated comparing the estimated

$\hat{m}$ and $\hat{p}$ with their real values m_0 and p_0 and considering the corresponding $\frac{\hat{p}}{\hat{m}}$ ratio, interpreted as a performance measure of the solution obtained running Algorithm 1 onto the decomposed constraint matrix $\tilde{A}$.

Tests are organized in the following sections as follows: Section 5.2 presents a sensitivity analysis of the procedure with respect to the input parameters $D_{\max}$ and m, Section 5.3 focuses on the computational time required by the procedure and, finally, Section 5.4 compares the two proposed strategies for estimating the number m of blocks and shows their effectiveness.

All simulations are carried out on a 6-core 2.6 GHz personal computer with 16 GB RAM.

5.2 Sensitivity analysis

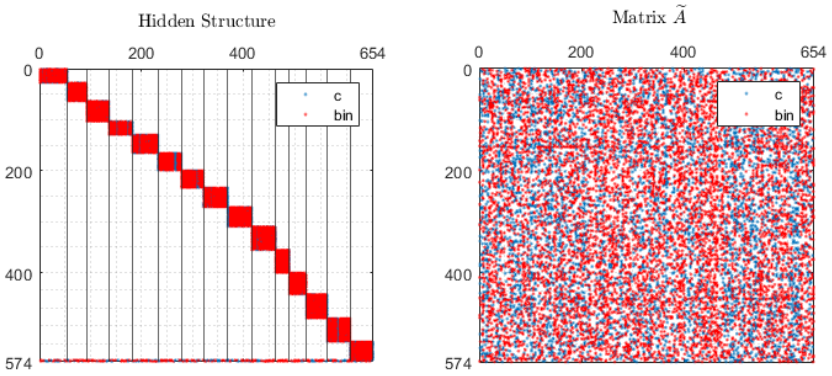
To assess the sensitivity of the proposed procedure with respect to the input parameters m and $D_{\max}$, we consider two scenarios: i) the hidden block-angular singly-bordered matrix structure has the same number of discrete decision variables in each block (*balanced* case), or ii) the blocks can have different number of discrete variables (*unbalanced* case); and then we experiment different values for m and $D_{\max}$ in both cases.

Notice that we explore different values of the parameter $D_{\max}$ to consider different computational capabilities for the agents.

Balanced distribution

We first consider running Algorithm 2 in the *ideal setting* where the constraint matrix $\tilde{A}$ is balanced, the parameter m is set to the true number of blocks m_0, and the maximum number $D_{\max}$ of discrete variables allowed per agent is set to $\frac{n_d}{m_0}$, where n_d is the total number of discrete variable.

In Figure 5.1 we report the hidden structure of the (balanced) constraint matrix used in the following tests (left) and its randomly permuted companion $\tilde{A}$ given as input to the proposed procedure. Non-zero elements

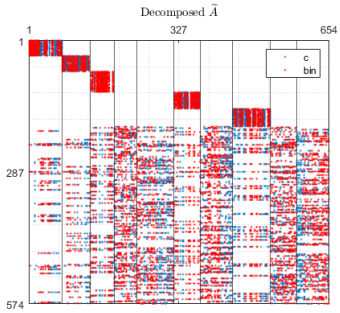


agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$n_{d,i}$	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24
$n_{c,i}$	29	16	20	19	21	28	19	20	19	24	11	13	15	21	19

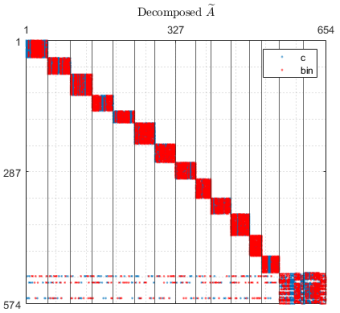
Figure 5.1: Original singly-bordered block-diagonal (left) and permuted version (right) of the balanced random matrix $\tilde{A}$ with the corresponding distribution of variables.

of the matrices are represented with dots and zero elements are not shown. The $n_d = 360$ discrete variables (red dots) are uniformly distributed among $m_0 = 15$ agents whilst continuous ones (blue dots) are randomly attributed to each part, so as to challenge the clustering operation. The table in Figure 5.1 reports the number $n_{d,i}$ of discrete decision variables and the number $n_{c,i}$ of continuous decision variables assigned to agent i, for all $i = 1, \ldots, m_0$. Finally, there are $p_0 = 3$ coupling constraints and, thus, the border-to-agents ratio $\frac{p_0}{m_0}$ is equal to $\frac{3}{15} = 0.2$.

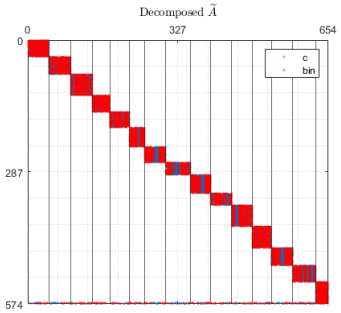
We run Algorithm 2 five times on the same input matrix setting $m = m_0 = 15$, but significantly over-estimating parameter $D_{\max} = 40$. The decomposed matrix $\widetilde{A}$ with the smallest ratio $\frac{\hat{p}}{\hat{m}} = \frac{386}{10}$ over the five runs, is reported in Figure 5.2a and, as can be seen, is far from being satisfactory. However, the presence of blocks with a number of discrete variables lower than the selected $D_{\max}$ (e.g. the forth block has 31 discrete



(a) $m = m_0, D_{\max} = 40 > n_{d,i}|_{\max}$.



(b) $m = m_0, D_{\max} = 35 > n_{d,i}|_{\max}$.



(c) $m = m_0, D_{\max} = 24 = n_{d,i}|_{\max}$.

(a)

agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$n_{d,i,0}$	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24
$n_{d,i,est}$	37	33	37	31	36	37	36	39	35	39					

(b)

agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$n_{d,i,0}$	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24
$n_{d,i,est}$	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24

(c)

agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$n_{d,i,0}$	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24
$n_{d,i,est}$	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24

Figure 5.2: Decomposed balanced sample matrix $\widetilde{A}$ in Figure 5.1 for $m = m_0$ and decreasing values of $D_{\max}$.

variables) suggests to reduce the parameter $D_{\max}$ to

$$D_{\max} = \frac{1}{10} \sum_{i=1}^{10} \hat{n}_{d,i} = 35 \tag{5.1}$$

where $\hat{n}_{d,i}$ denotes the number of discrete decision variables assigned to block i by Algorithm 2. Running again the procedure with $D_{\max} = 35$, the quality of the obtained decomposition improves. The best result over five runs, reported in Figure 5.2b, is characterised by a balanced distribution of the discrete variables, 24 per block, but fails in estimating the number of coupling constraints $\hat{p} = 67 \gg p_0$. However, from visual inspection it is possible to verify that only 3 of them are actually linking all the agents, whilst the others are coupling the blocks 14 and 15 only. The decomposition can be, then, further improved in two ways: running again the procedure considering the entire matrix and setting $D_{\max} = 24$, or considering the portion composed by the columns of the mentioned blocks. The obtained decompositions are shown in Figure 5.2c and Figure 5.3 respectively. Either ways, the algorithm is able to retrieve the hidden structure with $\frac{\hat{p}}{\hat{m}} = \frac{3}{15} = 0.2$.

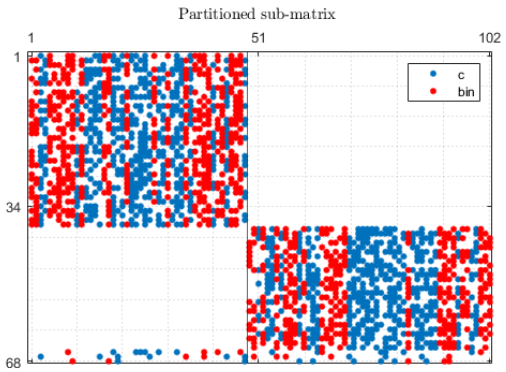


Figure 5.3: Result obtained applying the procedure to a submatrix obtained by the decomposed matrix in Figure 5.2b selecting only the columns of blocks $14, 15$ and setting $m = 2$,$D_{\max} = 24$.

The difficulties experienced by the procedure in case of over-estimation of $D_{\max}$ can be mainly attributed to fact that, during the random selection step of the sequential break-off procedure (Step 4 of Algorithm 2) large subsets of nodes of distinct agents may be grouped together. As a result,

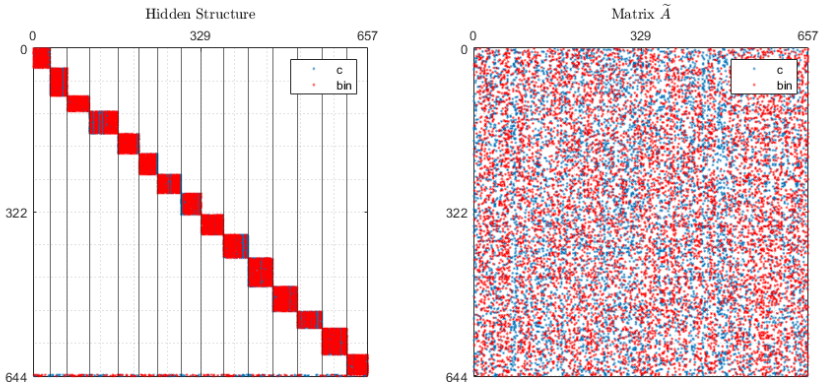
a long sequence of moves with negative gain may be needed to separate such nodes but - at the same time - many moves attributing others to the same part may yield a reduction of the cut-size. In similar conditions, penalisation in Algorithm 5 has little effect and the partitioning algorithm is likely to return a unsatisfactory solution. These situations can, however, be easily identified by a visual inspection of the result and a good estimate of $D_{\max}$ can be achieved with a couple of trials.

Unbalanced distribution

The case of unbalanced distribution of discrete variables among agents is typically more challenging for the procedure, which seeks for balanced partitions and hence requires the introduction of additional dummy nodes to handle unbalance. To provide a fair comparison, tests are carried out on a random matrix, the one depicted in Figure 5.4, with the same number of discrete variables ($n_d = 360$), blocks ($m_0 = 15$) and of coupling rows in the border ($p_0 = 3$) as in the tests on the balanced distribution case. In this unbalanced case, each block can contain between 20 and $n_{d,i}|_{\max} = 31$ discrete variables, but the overall number of discrete variables is still $n_d = 360$, as for the balanced case.

The procedure has satisfactory performance in the *ideal* configuration where $D_{\max}$ is set equal to the actual maximum number of discrete variables of the agents ($D_{\max} = \max_{i=1,\dots,m_0} \{n_{d,i}\} = n_{d,i}|_{\max}$) and the number of blocks m is set equal to the actual number of agents ($m = m_0$). An example is shown in Figure 5.5, where the true decomposition is retrieved despite the presence of agents that could be merged together without exceeding $D_{\max}$.

The true decomposition is instead only partially identified if one sets $D_{\max} = 25 < n_{d,i}|_{\max}$, see Figure 5.6a. However, also in this case, a visual inspection of the result can aid the selection of a better value for $D_{\max}$. Looking at the result in Figure 5.6a, 9 out of 15 blocks are clearly correctly isolated, thus allowing to identify rows that are likely to belong to the cut-set and many parts have a number of discrete variables close to the



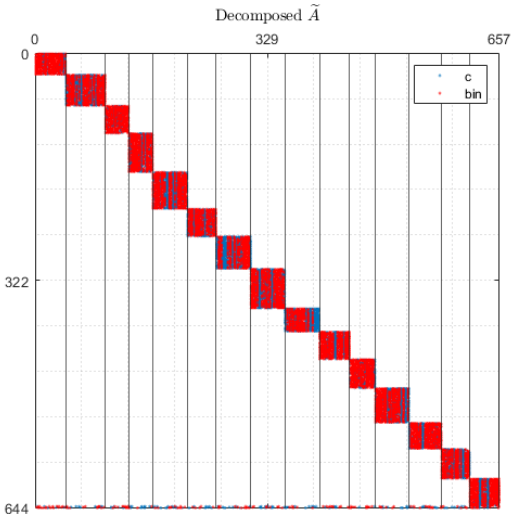
agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$n_{d,i}$	20	20	31	27	26	23	28	20	28	20	23	26	20	27	21
$n_{c,i}$	20	12	20	16	23	20	23	15	19	25	22	18	25	20	19

Figure 5.4: Original singly-bordered block-diagonal (left) and permuted version (right) of the unbalanced sample matrix $\tilde{A}$ with the corresponding distribution of variables.

allowed maximum, which suggests to increase parameter $D_{\max}$.

When $D_{\max}$ is overestimated, i.e., $D_{\max} = 40 > n_{d,i}|_{\max}$, the procedure tends to group different parts together. The resulting decomposed matrix is, in general, still satisfactory and its structure may allow identification of coupling constraints by visual inspection. Figure 5.6b shows an example of such a scenario with a re-arranged matrix counting $\hat{m} = 15$ blocks and $\hat{p} = 33$ rows in the border. In this case, the obtained decomposition may be further improved running again the procedure on the sub-matrix composed by the the columns of the coupled blocks. The estimated $\frac{\hat{p}}{\hat{m}}$ can also be reduced without even re-running the procedure, as considering blocks $14, 15$ as a unique block would reduce the coupling to $\hat{p} = p = 3$ at the price of increasing (but not exceeding) the computational load of the resulting aggregated agent (which will have $\hat{n}_{d,14} + \hat{n}_{d,15} = 18 + 13 = 31 < D_{\max}$ discrete variables).

If we now set $D_{\max} = 31$ and let m vary, tests show that the procedure

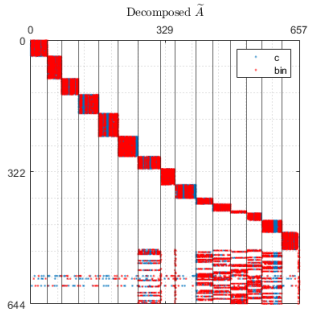


agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$n_{d,i,0}$	20	20	31	27	26	23	28	20	28	20	23	26	20	27	21
$n_{d,i,est}$	31	27	20	20	27	21	20	23	20	28	23	26	28	20	26

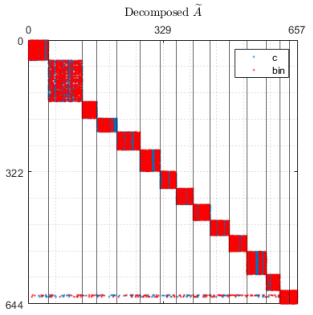
Figure 5.5: Decomposed matrix $\tilde{A}$ in the ideal setting.

performs better when the number of agents is overestimated rather than underestimated, due to the flexibility granted by the additional fictitious variables that allows to create empty groups and, hence, reduce the number of estimated parts with respect to the specified desired number of agents. Two examples are reported in Figures 5.6c and 5.6d for the choices $m = 13 < m_0$ and $m = 18 > m_0$, respectively. In both cases, the procedure fails at estimating the size of the border p ($\hat{p} = 140$, $\hat{p} = 33$) and the number of agents m ($\hat{m} = 13$, $\hat{m} = 16$), but it manages to reduce the number of agents from the user-specified $m = 18$ to the estimated $\hat{m} = 16$ in case of over-estimation. In addition, the resulting decomposition has a lower $\frac{\hat{p}}{\hat{m}}$ ratio and allows to identify the hidden structure by visual inspection (blocks $15, 16$ can be aggregated compatibly with $D_{\max}$), similarly to the previous case.

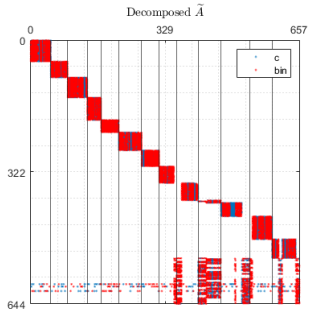
Note that performance may worsen in case of unfortunate combinations



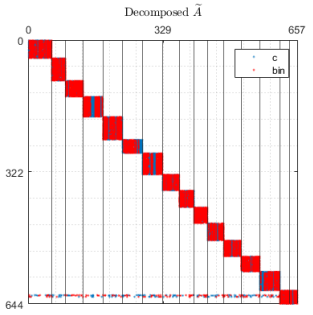
(a) $m = m_0, D_{\max} = 25 < n_{d,i}|_{\max}$.



(b) $m = m_0, D_{\max} = 40 > n_{d,i}|_{\max}$.



(c) $m = 13 < m_0, D_{\max} = 31$.



(d) $m = 18 > m_0, D_{\max} = 31$.

	agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(a)	$n_{d,i,0}$	20	20	31	27	26	23	28	20	28	20	23	26	20	27	21
	$n_{d,i,est}$	21	20	26	20	23	26	26	22	23	26	26	26	26	26	23
	agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(b)	$n_{d,i,0}$	20	20	31	27	26	23	28	20	28	20	23	26	20	27	21
	$n_{d,i,est}$	26	40	23	20	27	27	20	21	26	28	28	23	20	18	13
	agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(c)	$n_{d,i,0}$	20	20	31	27	26	23	28	20	28	20	23	26	20	27	21
	$n_{d,i,est}$	20	26	27	28	31	20	23	29	32	31	32	32	29		
	agent	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(d)	$n_{d,i,0}$	20	20	31	27	26	23	28	20	28	20	23	26	20	27	21	
	$n_{d,i,est}$	27	20	28	26	23	20	27	21	23	20	20	26	28	20	19	12

Figure 5.6: Decomposed $\tilde{A}$ with different settings and relative distributions

of m and $D_{\max}$, depending on the properties of the decomposed matrix. Providing a complete overview of the all the possible tuning choices is outside the scope of this analysis.

To show the effectiveness of the procedure in addressing large-scale MILPs decomposition, we perform tests with large number of parts m_0 and number of decision variables. Figure 5.7 shows a sample matrix $\tilde{A}$ hiding a multi-agent structure of $m_0 = 80$ agents coupled by $p_0 = 16$ constraints, and characterised by an unbalanced distribution of discrete variables with $\epsilon_0 = 0.2632$.The obtained decomposition is shown in Figure 5.8. All the constraints in the border are correctly identified ($\hat{p} = 16$), but two pairs of distinct agents are aggregated leading to an estimated $\hat{m} = 78 < m_0 = 80$ and a corresponding 2% relative increase in the border-to-agents ratio.

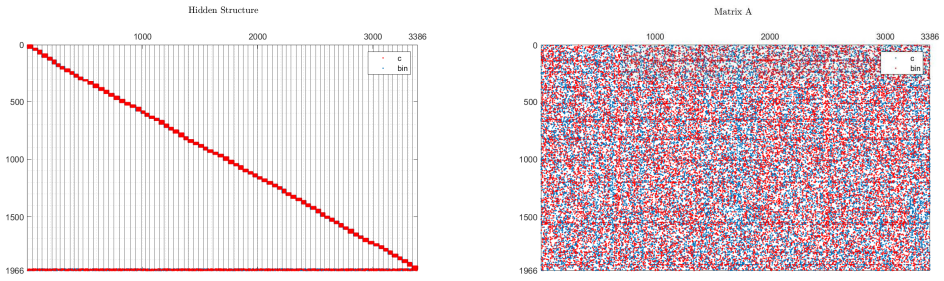


Figure 5.7: Original singly-bordered block-diagonal (left) and permuted version (right) of a sample matrix $\tilde{A}$ with $m = 80$ agents, in the unbalanced case.

5.3 Computational time

In order to determine the time needed for the hyper-graph partitioning Algorithm 2 to find a singly-bordered block-angular decomposition of a matrix $\tilde{A}$, we have to account for the contribution of the following operations:

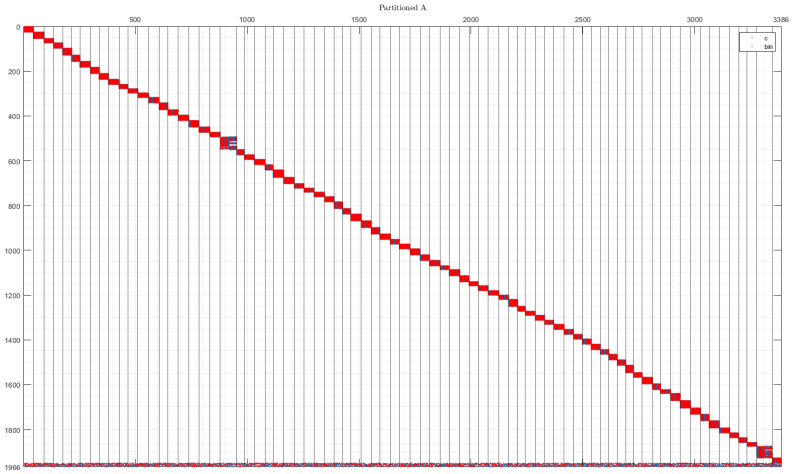


Figure 5.8: Decomposed matrix obtained from input matrix in Figure 5.7: $\hat{m} = 78$ vs $m_0 = 80$ and $\frac{\hat{p}}{\hat{m}} = 0.2051$ vs $\frac{p_0}{m_0} = 0.2$

- a preliminary *clustering* that associates the continuous nodes to the discrete ones and creates a coarsened hyper-graph
- the computation of a *h-way partition* of such a hyper-graph
- an *uncoarsening* step that expresses the obtained partition in terms of the nodes of the initial uncoarsened hyper-graph
- a block-isolation procedure that identifies the parts to be stored in the final partition and reduces the hyper-graph accordingly

The plot in Figure 5.10 summarizes information on the impact that each phase has on the run-time of a single iteration and refers to the decomposition of the sample matrix $\tilde{A}$ in Figure 5.9. Most of the computational effort is concentrated in the partitioning phase, that takes more than 95% of the time in the first and second iterations and drops below 90% at the third one. As expected, the partitioning phase has a contribution on the total execution time per iteration that decreases as more parts are identified and removed from the hyper-graph due to the isolation step; however, it remains the operation with the main impact on the overall run-time.

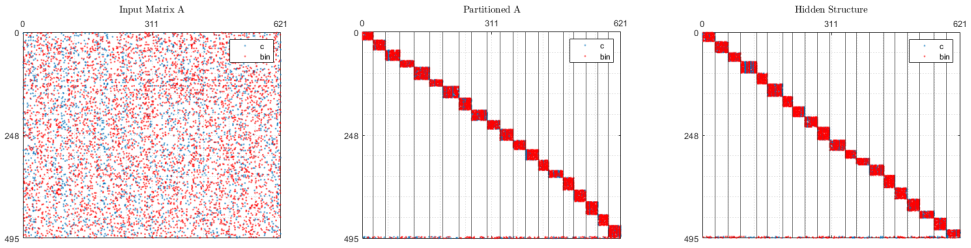
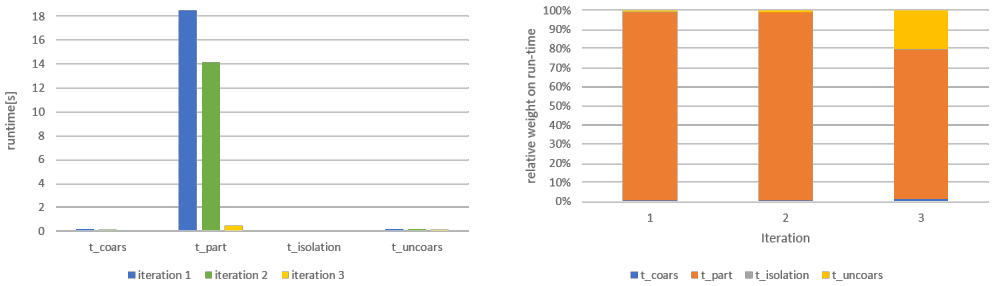


Figure 5.9: Decomposition of a balanced sample matrix with $m_0 = 17$. From left to right: matrix A, decomposed matrix and hidden structure



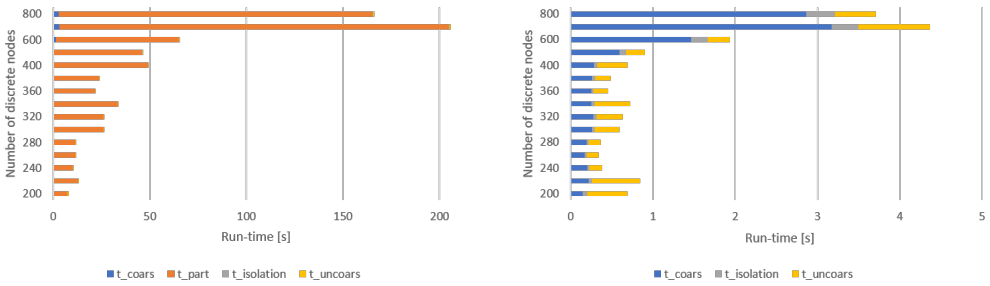
(a) absolute run-time per iteration

(b) relative impact per iteration

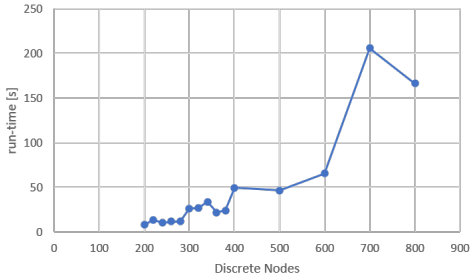
Figure 5.10: Plot of the absolute and relative run-time of each of the four main operations of the procedure per iteration.

As a consequence, it is the number of discrete decision variables in a MILP to determine the computational effort involved in its reformulation as multi-agent constraint-coupled MILP, since it coincides with the number of supernodes in the coarsened hyper-graph to be partitioned. The number of continuous decision variables, instead, has little impact since it affects the run-time of the coarsening and uncoarsening steps, respectively represented in blue and yellow in Figure 5.10. Note that also the number of parts m plays an important role, since it coincides with the worst-case number of iterations of the coarsening-partitioning-isolation procedure.

To investigate the relationship between the number of discrete nodes and the overall computational time, we test the procedure on matrices $\tilde{A}$ of increasing dimensions with underlying singly-bordered block-angular



(a) run-time including partitioning **(b)** run-time without partitioning



(c) total run-time vs number of discrete nodes

Figure 5.11: Assessment of the dependence of the run-time on the number of discrete decision variables.

structure composed of $m_0 = 17$ blocks, containing $\{n_{d,i}\}_{i=1}^{17} = 20$ discrete variables each, and characterised by a border-to-agents ratio equal to $\frac{p_0}{m_0} = 0.2$. We perform 5 repetitions with different seeds for each test instance and report the arithmetic mean of the cumulative run-time (in seconds) of the four operations across the partition. Figure 5.11a reports values for all coarsening (blue), partitioning (orange), uncoarsening (yellow) and isolation (grey) phases, whilst Figure 5.11b provides the same plot, neglecting the partitioning one.

Figure 5.11c shows that the total run-time increases more than linearly with the number of discrete decision variables. This is mainly due to the gain update performed in the FM procedure, that has a worst-case complexity per pass growing quadratically with the sum of pins and nets $P = \sum_{n\in\mathcal{N}}|\text{Pins}(n)| + \sum_{u\in\mathcal{U}}|\text{Nets}(u)|$ in the hyper-graph to be partitioned

$\mathcal{H} = (\mathcal{U}, \mathcal{N})$ (see [35] for mathematical derivation). Since a new pass is performed only if the previous on reduces the cut-size, the worst-case execution is the one that produces a reduction $\Delta_{cs} = -1$ at any pass and that, hence, performs a worst-case number of passes equal to $|\mathcal{N}|$. Ultimately, the sole Step 7 of Algorithm 4 requires a number $O(|\mathcal{N}|P^2)$ of operations.

5.4 Identification of the number of blocks

All tests presented consider a configuration in which at least one between $D_{\max}$ and m is correctly set. In real applications, however, one typically has little or no insight on the hidden structure to be recovered and it is, hence, likely to choose incorrect values for both parameters. The integration of Algorithm 2 with one of the suggested approaches for the estimation of the number of agents allows the user to focus the tuning on parameter $D_{\max}$ whilst the procedure provides an estimated for m.

The first identification strategy proposed in Section 4.3.6 estimates the number of blocks m_0 of the equivalent constraint-coupled multi-agent MILP reformulation with an iterative approach, trying different candidate values in an ordered sequence. To assess its effectiveness we test the procedure considering both structures having balanced and unbalanced distribution of discrete decision variables among the agents. Note that, since simulations performed in Section 5.2 highlight a robustness of the procedure with respect to an over-estimation of the number of agents ($m > m_0$), the choice of candidate m follows a *top-down* approach, starting from $m = \left\lceil \frac{n_d}{D_{\min}} \right\rceil$ and possibly reaching $m = \left\lfloor \frac{n_d}{D_{\max}} \right\rfloor$, where $D_{\min}$ and $D_{\max}$ respectively represent the maximum and minimum number of discrete variables per agent.

We first test the procedure on a matrix hiding a structure composed of $m_0 = 18$ parts with $\{n_{d,i}\}_{i=1}^{18} = 20$ discrete decision variables each, for a total of $n_d = 360$ discrete decision variables. Values $D_{\max}$ and $D_{\min}$ are

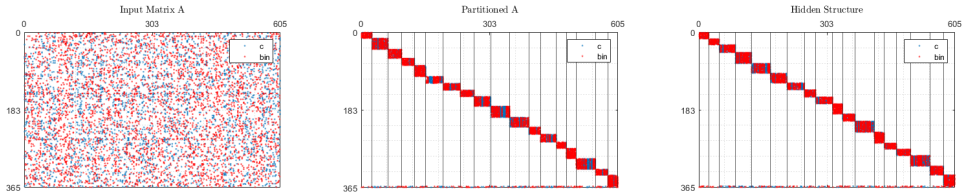


Figure 5.12: Outcome of the iterative m-identification strategy applied to a balanced distribution of discrete variables.From left to right: matrix $\tilde{A}$, decomposed matrix and hidden structure.

set to $D_{\max} = 25$ and $D_{\min} = 10$. Figure 5.12 shows the input matrix $\tilde{A}$ (left), the decomposed matrix (centre) and the hidden structure (right). The procedure considers as first candidate the value $m = \left\lceil \frac{n_d}{D_{\min}} = 36 \right\rceil$ and retrieves a singly-bordered block-angular decomposition having $\hat{m} = 19$ blocks with a border-to-agents ratio equal to $\frac{\hat{p}}{\hat{m}} = 1,4211$. It, then, runs again the procedure setting $m = 19$ and further reduces the number of blocks to $\hat{m} = 18$ and the border-to-agents ratio $\frac{\hat{p}}{\hat{m}} = 0.1667$. An additional iteration is performed, but no improvement in the border-to-agents ratio is obtained and the routine stops and returns the best obtained decomposition, corresponding to the iteration $m = 19$ and reported in Figure 5.12. The evolution is summarized in Figure 5.13.

We then perform a test considering an unbalanced structure having the same number of blocks m_0, with $n_{d,i}|_{\max} = 20$ and relative unbalance of $\epsilon_0 = 0.667$. Following a *top-down* approach, the first run of the procedure considers as first tentative value $m = \left\lceil \frac{n_d}{D_{\min}} \right\rceil = \frac{203}{10} = 21$ and retrieves a decomposition having $\hat{m} = 12$ blocks with border-to-agents ratio $\frac{\hat{p}}{\hat{m}} = \frac{3}{12} = 0.25$ and an estimated relative unbalance $\hat{\varepsilon} = 0.5294$. It then applies the procedure again setting $m = 12$ but no improvement is found and the m-identification procedure stops. Figure 5.14 shows the input matrix and the hidden singly-bordered block-angular structure, whereas Figure 5.15a and 5.15b report the two decompositions found, having equal estimated border-to-agent ratio $\frac{\hat{p}}{\hat{m}} = \frac{3}{12} = 0.25$ but different relative unbalance

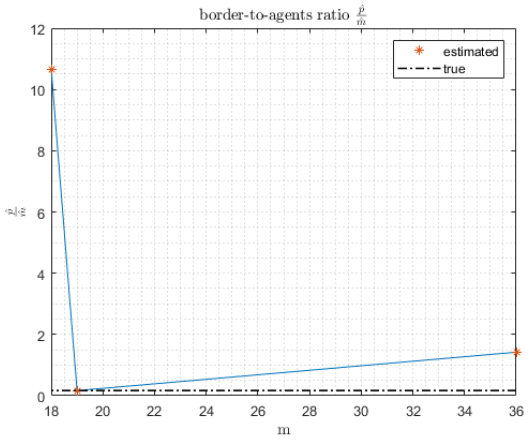


Figure 5.13: Evolution of the estimated border-to-agent ratio $\frac{\hat{p}}{\hat{m}}$ during the test of the iterative m-identification strategy with balanced distribution of discrete variables.

$\hat{\varepsilon}_1 = 0.5294$ and $\hat{\varepsilon}_1 = 0.4118$ respectively.

Note that, due to the unbalanced distribution, the procedure finds a minimum cut-size partitioned but fails in estimating the number of agents m_0, thus obtaining a higher $\frac{p}{m}$ ratio.

Construction of a k-ary tree

The second approach proposed in Section 4.3.6 to estimate the number of blocks m_0 recursively applies the partitioning algorithm to the hyper-graph representing matrix $\tilde{A}$ and the obtained sub-hyper-graphs, choosing tentative values in the set $\mathcal{C}_K$, collecting the first K prime numbers.

We run the procedure on the test instance in Figure 5.16 with hidden structure composed of $m_0 = 12$ blocks coupled by $p_0 = 2$ coupling constraints, characterised by a balanced distribution of the discrete variables, which are set equal to 20 per agent. We set $m_{target} = 12$, $\pi = 0.5$ and choose to iterate over $k \in \mathcal{C} = \{2, 3, 5, 7\}$. The obtained tree of decompositions is depicted in Figure 5.17:

- grey leaf nodes are associated to unsatisfactory partitions for which

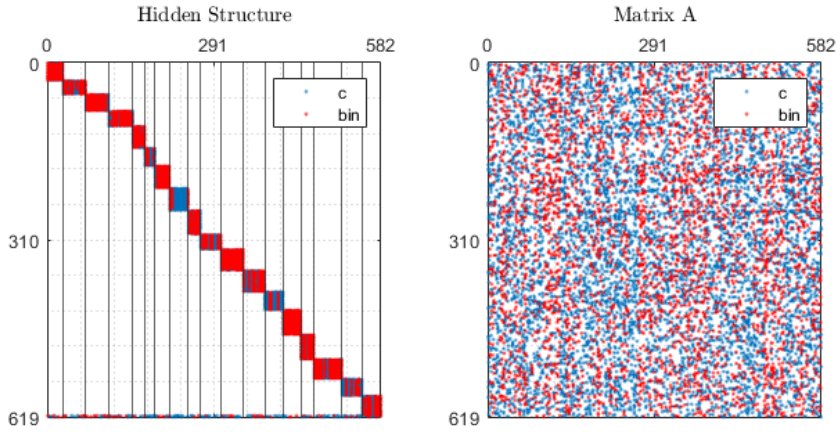


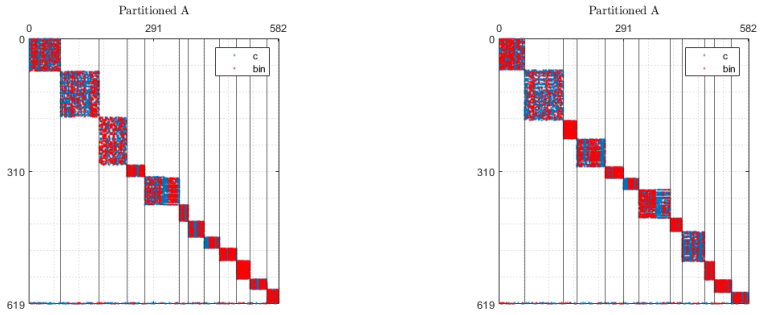
Figure 5.14: Hidden structure (left) and input matrix $\tilde{A}$ (right) used in the second test of the m-identification iterative routine.

the estimated $\frac{\hat{p}_\nu}{m_{target}}$ is greater than the threshold π.

- yellow leaf nodes correspond to good partitions for which the estimated $\frac{\hat{p}}{m_{target}}$ is below the threshold and equal to the smallest upper-bound found.

Nodes associated with unsatisfactory partitions at level 4 of the tree are not reported for the sake of compactness and readability.

The tree must be interpreted as follows: each node ν contains an ordered sequence denoting the number of parts in which the corresponding sub-hyper-graphs in the father node are divided. Figure 5.18 provides two examples of interpretation of a given path on the tree. Path a contains the orange branches: the hyper-graph corresponding to the initial matrix is first divided in 2, each induced sub-hyper-graph is again bisected when going from level 2 to level 3 and finally, the first 2 sub-hyper-graphs of the four obtained at level 3 are partitioned in 3 parts each. Path b contains the yellow branches: the initial hyper-graph is divided in 2, each half is, then, partitioned in 3 and finally the sub-hyper-graphs are further divided in 2.



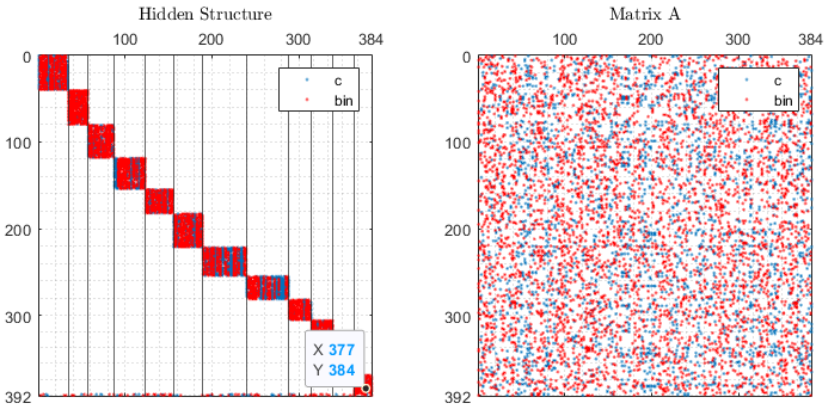
(a) Iteration 1, $m = 21$ **(b)** Iteration 2, $m = 12$

Figure 5.15: Decomposed matrices obtained during the iterative m - identification tested on the matrix in Figure 5.14.

A complete decomposition is obtained starting from leaf nodes associated to satisfactory partitions (the yellow ones) having the same father node. The test run finds 2 decomposition for matrix $\tilde{A}$. The first is obtained dividing the initial hyper-graph in 2, then again each half in 2 and finally each quarter in 3. The second, instead, is obtained taking an alternative branch at level 2: the hyper-graph is bisected, each half is divided in 3 and each third is finally bisected in 2. Both paths yield to a minimum cut-size 12-way partition of the original hyper-graph, associated to a singly-bordered block-angular decomposition of the input matrix $\tilde{A}$ composed of $\hat{m} = 12 = m_0$ blocks and $\hat{p} = 2 = p_0$ rows in the border.

Note that node ν_3 is not further divided, despite the fact that the correct number of parts $m = 12$ can be obtained partitioning the hyper-graph in 3 parts, and then bisecting each sub-hypergraph twice. The algorithms fails at finding a satisfactory 3-way partition and hence the branch is pruned due to a large $\frac{\hat{p}}{\hat{m}}$ ratio.

In order to compare the two strategies for estimating the number of blocks m, we run also the iterative routine exploring an ordered sequence of candidate m values on matrix in Figure 5.16 setting $D_{\max} = 25$ and $D_{\min} = 10$. Figure 5.19 shows the final decompositions obtained with the iterative routine (a) and the exploration of the tree (b). Both procedures



agent	1	2	3	4	5	6	7	8	9	10	11	12	13
$n_{d,i,0}$	20	20	20	20	20	20	20	20	20	20	20	20	

Figure 5.16: Original singly-bordered block-diagonal (left) and permuted version (right) of the balanced sample matrix $\tilde{A}$ considered to test the m-identification routines, with the corresponding distribution of variables.

return a final decomposition counting $\hat{m} = 12$ blocks, and $\hat{p} = 2$ rows in the border.

The choice of the strategy depends on the problem at hand and the number of discrete decision variables, in particular. Building the tree typically requires computing many unsatisfactory partition, but allows to reduce the dimensions of the hyper-graph and the number of blocks with a reduction of the run-time of the partitioning operation. The iterative method, instead, applies repeatedly the partitioning procedure on the same matrix trying different values for m, but is likely to perform a smaller number of trials, thus running the partitioning algorithm a smaller number of times.

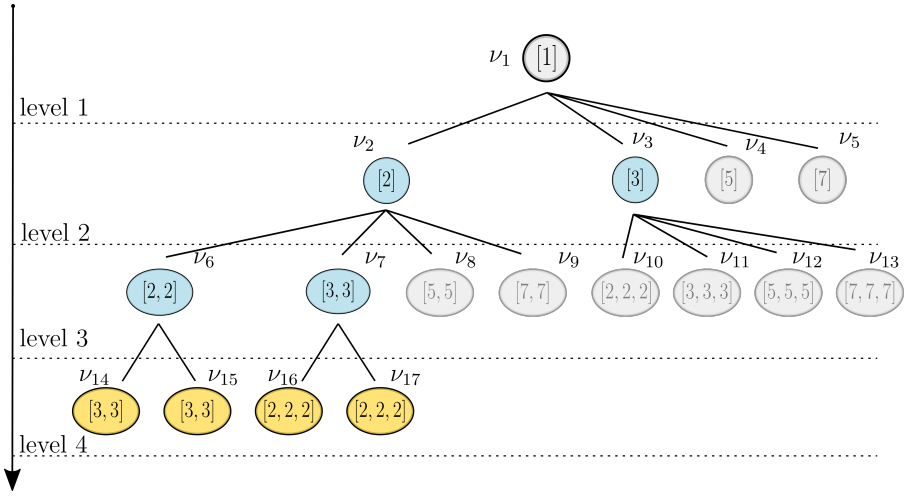


Figure 5.17: Tree built during the m identification routine while partitioning the sample matrix in Figure 5.16.

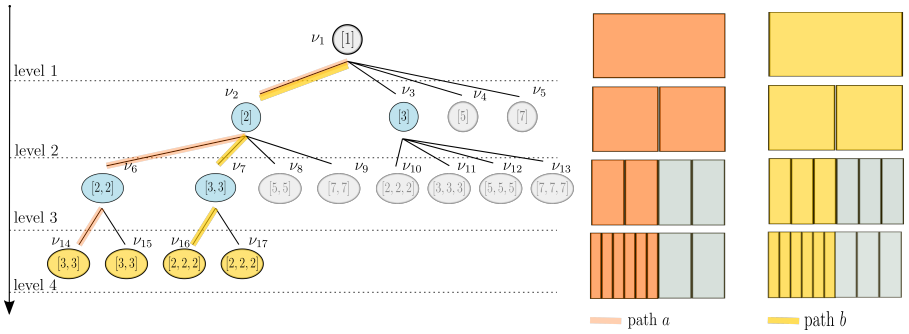
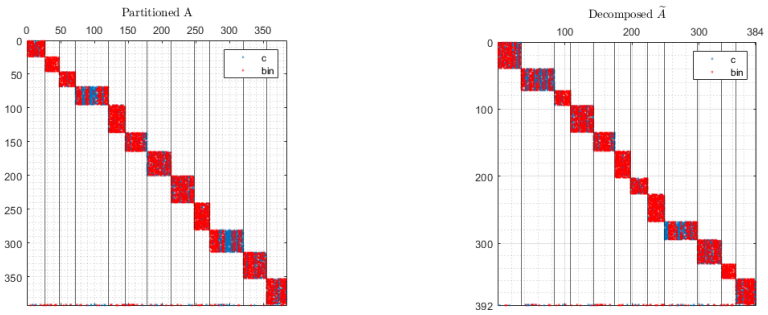


Figure 5.18: Interpretation of a path of the tree. Tree (left) and visual representation of the partitioning operations.



(a) **(b)**

Figure 5.19: Outcome of the two m-identification strategies applied to the balanced distribution of discrete variables in Figure 5.16

Chapter 6

Conclusions and future work

6.1 Conclusions

This work addressed the problem of finding a constraint-coupled multi-agent reformulation for a generic large-scale MILP, so as to make it suited for the resolution via distributed optimization algorithms.

We first pointed out the relevance of the problem, since MILPs can model various engineering applications of interest, but often have a large size and are then hard to solve in practice. We then provided an overview of the existing optimal and heuristic resolution schemes for generic MILPs, and described a recently proposed decentralized optimization algorithm tailored to constraint-coupled multi-agent MILPs, where a feasible solution with performance guarantees is computed in a finite number of iterations. Scalability is recovered by making agents solve in parallel lower-dimensional MILPs associated to their own decision variables, while exchanging information on the coupling constraints.

In an effort of making it a general-purpose tool for the resolution of large-scale MILPs, we proposed a two-step strategy that first discloses an equivalent constraint-coupled multi-agent structure of the MILP and then retrieves a solution applying the mentioned decentralized algorithm. Key to the effectiveness of the approach is the even distribution of discrete

decision variables among the fictitious agents, and the minimization of the coupling constraints.

We then focused on the decomposition step showing how the desired reformulation can be obtained by reducing the matrix defining the linear constraints in the MILP to a singly-bordered block-angular structure (with blocks corresponding to agents and border to coupling constraints) by means of permutation of its rows and columns. After translating the matrix manipulation problem into a hyper-graph partitioning one, we introduced a novel algorithm inspired to state-of-the-art partitioning tools but with innovative elements to account for the specific requirements of our settings. In particular, we designed

- a preliminary clustering phase that aggregates nodes corresponding to the continuous and discrete decision variables so as to evenly distribute the discrete ones while partitioning the resulting coarsened hyper-graph;

- a refinement strategy based on the iterative identification and isolation of the different agents that allows to exploit randomization of the coarsening operation to improve a partition while reducing the size of the hyper-graph.

We also suggested two different approaches to estimate the number of fictitious agents in the reformulation, to be maximized while minimizing the number of coupling constraints. A first iterative routine starts from an initial guess for the number of agents and iteratively reduces it (or increases it, depending on whether a top-down or bottom-up paradigm is followed) until no improvement in the final partition is obtained. An alternative approach, instead, creates a tree of partitions by recursively applying the procedure on the initial hyper-graph and the obtained sub-hyper-graphs, choosing prime numbers as candidate values for the number of blocks at every stage in the tree construction.

Performance of the novel partitioning strategy was assessed in simu-

lation, considering matrices with a hidden singly-bordered block-angular structure generated at random. The introduction of fictitious nodes in the hyper-graph enabled the procedure to retrieve the correct matrix reformulation in case of both balanced and unbalanced hidden distribution of the discrete variables, and for different values of the user-defined parameters. Tests were also instrumental to give guidelines on how to modify such parameters to get satisfactory results. Finally, we compared the two strategies proposed to identify (and maximize) the number of agents in the final reformulation.

The proposed procedure gave promising results when applied to matrices with underlying singly-bordered block angular structure.

6.2 Future work

Interesting research directions can be explored to improve the proposed partitioning algorithm, as discussed below.

A further more challenging topic is how to handle the case when the MILP has not 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 structure. 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 thesis work.

Recursive Bisection

The first extension concerns an alternative implementation of the m-way partitioning algorithm, in which the sequential break-off paradigm is replaced by a recursive bisection procedure. As explained in Section 4.4, the standard recursive strategy does not account for the limitation on the maximum number of discrete variables allowed per agent (and hence of supernodes admitted per part) and, hence, needs to be modified accordingly to fit the considered set-up. This could be done by introducing an adaptive

balance parameter, but the design of the law describing its evolution needs to be further investigated.

Note that the introduction of a recursive bisection paradigm would pave the way to the implementation of multilevel refinement strategies (see Section 4.4), to further improve the obtained partition.

Estimate the border-to-agents ratio within the m-identification tree

Another interesting research direction focuses on the exploration of the tree of partitions created by one of the strategies proposed in Section 4.3.6 for the estimation of the number of agents. More in detail, the current implementation evaluates the border-to-agents ratio $\frac{p}{m}$ at each node of the tree setting $m = m_{target}$, with m_{target} defined *a-priori*, thus causing the pruning decision to be based on the size p of the border only. The introduction of a technique to estimate the final number of parts in the partition based on the information available at each node can be crucial in order to improve the effectiveness of the pruning operation and, hence, reduce the unnecessary partitions computed by the procedure.

Appendix A

MLD operating constraints

The final objective is rewriting the constraints so as to obtain a structure as the following:

$$0 \leq E_{1t} \begin{bmatrix} P_1^G(t) \\ P_1^B(t) \\ \delta_1^G(t) \\ \delta_1^L(t) \\ \vdots \\ P_m^G(t) \\ P_m^B(t) \\ \delta_m^L(t) \end{bmatrix} + E_{4t} \begin{bmatrix} S_1(t) \\ \vdots \\ S_N(t) \end{bmatrix} + E_{5t} \tag{A.1}$$

In the sequel, the reformulation is carried on focusing on the i^th component of vectors $s(t), u(t), z(t)$, hence deriving the i^th column block of the compact matricial form.

Min/Max Energy Level

$$\underline{S}_i \leq S_i(t) \leq \overline{S}_i \quad \rightarrow \quad \begin{cases} 0 \leq +S_i(t) - \underline{S}_i \\ 0 \leq -S_i(t) + \overline{S}_i \end{cases}$$

$$0_{2\times 1} \leq 0_{2\times(3+J_i^L)} u_i(t) + \begin{bmatrix} +1 \\ -1 \end{bmatrix} s_i(t) + \begin{bmatrix} -\underline{S}_i \\ +\overline{S}_i \end{bmatrix} \tag{A.2}$$

Discharging/Charging Rates Limitation

$$P_i^{B,c} \leq P_i^B(t) \leq P_i^{B,d} \quad \rightarrow \quad \begin{cases} 0 \leq +P_i^B(t) - P_i^{B,c} \\ 0 \leq -P_i^B(t) + P_i^{B,d} \end{cases}$$

$$0_{2\times 1} \leq \begin{bmatrix} 0 & 1 & 0 & 0_{1\times J_i^L} \\ 0 & -1 & 0 & 0_{1\times J_i^L} \end{bmatrix} u_i(t) + \begin{bmatrix} 0 \\ 0 \end{bmatrix} s_i(t) + \begin{bmatrix} -P_i^{B,c} \\ +P_i^{B,d} \end{bmatrix} \tag{A.3}$$

Min/Max Power Produced

$$\delta_i^G(t)\underline{P}_i^G \leq P_i^G(t) \leq \delta_i^G(t)\overline{P}_i^G \quad \rightarrow \quad \begin{cases} 0 \leq +P_i^G(t) - \delta_i^G(t)\underline{P}_i^G \\ 0 \leq -P_i^G(t) + \delta_i^G(t)\overline{P}_i^G \end{cases}$$

$$0_{2\times 1} \leq \begin{bmatrix} +1 & 0 & -1 & 0_{1\times J_i^L} \\ -1 & 0 & +1 & 0_{1\times J_i^L} \end{bmatrix} u_i(t) + \begin{bmatrix} 0 \\ 0 \end{bmatrix} s_i(t) + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{A.4}$$

Power Consumption by L

$$P_i^L(t) = \frac{\overline{P}_i^L}{n_i^L} \sum_{j=1}^{J_i^L} \left(2^{j-1} \cdot \delta_{i,j}^L(t)\right) = \overbrace{\begin{bmatrix} 1 & 2 & \cdots & 2^{J_i^L-1} \end{bmatrix}}^{\sigma_i^\top} \overbrace{\begin{bmatrix} \delta_{i,1}^L(t) \\ \vdots \\ \delta_{i,J_i^L}^L(t) \end{bmatrix}}^{\delta_i^L(t)} = \sigma_i^\top \delta_i^L(t)$$

Such constraint is not required to be expressed via inequalities, being it is implicitly enforced replacing $P_i^L(t)$ with the corresponding expression when necessary.

Correctness of the amount of energy supplied to L

$$E_i^L = \sum_{t=t_0}^{M-1} \Delta_t s P_i^L(t) \quad \rightarrow \quad \begin{cases} 0 \leq -\Delta t_s \sum_{t=t_0}^{M-1} P_i^L + E_i^L \\ 0 \leq +\Delta t_s \sum_{t=t_0}^{M-1} P_i^L - E_i^L \end{cases}$$

Defining:

- $\eta_{u,i} = \begin{bmatrix} 0 & 0 & 0 & \sigma_i^\top \end{bmatrix} \in \mathbb{R}^{(J_i^L+3)}$
- $\zeta_{u,i} = \begin{bmatrix} 0 & 0 & 0 & 0_{1\times J_i^L} \end{bmatrix} \in \mathbb{R}^{J_i^L\times(J_i^L+3)}$

constraint (2.11) can be expressed in compact notation as:

$$0_{2\times 1} \leq \Delta t_s \cdot \begin{bmatrix} -1_{1\times M-t_0} \\ +1_{1\times M-t_0} \end{bmatrix} \begin{bmatrix} P_i^L(t_0) \\ \vdots \\ P_i^L(M-1) \end{bmatrix}$$

$$0_{2\times 1} \leq \Delta t_s \cdot \begin{bmatrix} -1_{1\times M-t_0} \\ +1_{1\times M-t_0} \end{bmatrix} \begin{bmatrix} \eta_{u,i} & \zeta_{u,i} & \cdots & \zeta_{u,i} \\ \zeta_{u,i} & \eta_{u,i} & \ddots & \zeta_{u,i} \\ \vdots & \ddots & \ddots & \vdots \\ \zeta_{u,i} & \zeta_{u,i} & \cdots & \eta_{u,i} \end{bmatrix} \begin{bmatrix} u_i(t_0) \\ \vdots \\ u_i(M-1) \end{bmatrix} \quad \text{(A.5)}$$

Notice that the form of (A.5) is different from the one required in (A.1) because the constraint expressed involves values of vector $u(t)$ at different time instants.

Appendix B

MILP constraints

The following appendix handles the MILP constraints and shows how to derive the compact notation

$$\sum_{i=1}^{N} A_i x_i \leq b$$
$$x_i \in X_i, i = 1, ..., N$$

We start by recalling that

$$x_i^\top(t) = \left[u_i(t_0)^\top \, h_i^B(t_0)^\top \, h_i^L(t_0)^\top \, \dots \, u_i(M-1)^\top \, h_i^B(M-1)^\top \, h_i^L(M-1)^\top\right]$$
$$u_i^\top(t) = \left[P_i^G(t)^\top \, P_i^B(t)^\top \, \delta_i^G(t)^\top \, \delta_i^L(t)^\top\right]$$
$$\delta_i^L(t)^\top = \left[\delta_{i,1}^L(t)^\top \, \dots \, \delta_{i,J_i^L}^L(t)^\top\right]$$

Local Constraints

The condition $x_i \in X_i$, where X_i is a mixed integral polyhedral set, can be expressed as $D_i \cdot x_i \leq d_i$. We next retrieve the explicit expression of matrixes D_i, that define the block-diagonal submatrix of the overall constraint one.

Battery Storage Dynamics

$$S_i(t+1) = S_i(0) - \Delta t_s \sum_{s=0}^{t} P_i^B(s) = S_i(t) - \Delta t_s P_i^B(t)$$

Implicitly enforced replacing the expression for $S_i(t)$ when needed.

Min/Max Energy Level

$$\underline{S}_i \leq S_i(t) \leq \overline{S}_i$$

$$\rightarrow \quad \begin{cases} +\Delta t_S \sum_{k=t_0}^{t-1} P_i^B(k) \leq +S_i(t_0) - \underline{S}_i \\ -\Delta t_S \sum_{k=t_0}^{t-1} P_i^B(k) \leq -S_i(t_0) + \overline{S}_i \end{cases} \quad \forall t = t_0 + 1, ..., M-1$$

Defining:

- $\zeta_{xi} = 0_{1\times(J_i^L+5)}$
- $v_i^{EL} = \begin{bmatrix} 0 & \Delta t_s & 0 & 0_{1\times J_i^L} & 0 & 0 \end{bmatrix} \in \mathbb{R}^{J_i^L+5}$

constraints (2.8) can be rewritten as

$$\begin{bmatrix} v_1^{EL} & \zeta_{xi} & \cdots & \zeta_{xi} \\ v_1^{EL} & v_2^{EL} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ v_1^{EL} & v_2^{EL} & \cdots & v_m^{EL} \\ -v_1^{EL} & \zeta_{xi} & \cdots & \zeta_{xi} \\ -v_1^{EL} & -v_2^{EL} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ -v_1^{EL} & -v_2^{EL} & \cdots & -v_m^{EL} \end{bmatrix} x_i \leq \begin{bmatrix} S_i(t_0) - \underline{S}_i \\ \vdots \\ S_i(t_0) - \underline{S}_i \\ -S_i(t_0) + \overline{S}_i \\ \vdots \\ -S_i(t_0) + \overline{S}_i \end{bmatrix}$$

Discharging/Charging rates limitation

$$P_i^{B,c} \leq P_i^B(t) \leq P_i^{B,d} \quad \rightarrow \quad \begin{cases} -P_i^B(t) \leq -P_i^{B,c} \\ +P_i^B(t) \leq P_i^{B,d} \end{cases} \quad \forall t = t_0, ..., M-1$$

Defining

$$v_{DCL} = \begin{bmatrix} 0 & 1 & 0 & 0_{1\times J_i^L} & 0 & 0 \end{bmatrix} \in \mathbb{R}^{1\times(J_i^L+5)}$$

the $M - t_0$ constraints can be rewritten as

$$\begin{bmatrix} -v_{DCL} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & -v_{DCL} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \zeta_{xi} & \cdots & -v_{DCL} \\ v_{DCL} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & v_{DCL} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \zeta_{xi} & \cdots & v_{DCL} \end{bmatrix} x_i \leq \begin{bmatrix} -P_i^{B,c} \\ -P_i^{B,c} \\ \vdots \\ -P_i^{B,c} \\ P_i^{B,d} \\ P_i^{B,d} \\ \vdots \\ P_i^{B,d} \end{bmatrix}$$

Min/Max power produced by G

$$\delta_i^G(t)\underline{P}_i^G \leq P_i^G(t) \leq \delta_i^G(t)\overline{P}_i^G$$

$$\rightarrow \quad \begin{cases} -P_i^G(t) + \delta_i^G(t)\underline{P}_i^G \leq 0 \\ +P_i^G(t) - \delta_i^G(t)\overline{P}_i^G \leq 0 \end{cases} \quad \forall t = t_0, ..., M-1$$

Defined:

- $v_{PG} = \begin{bmatrix} -1 & 0 & \underline{P}_i^G & 0_{1\times J_i^L} & 0 & 0 \end{bmatrix} \in \mathbb{R}^{1\times(J_i^L+5)}$
- $w_{PG} = \begin{bmatrix} +1 & 0 & \overline{P}_i^G & 0_{1\times J_i^L} & 0 & 0 \end{bmatrix} \in \mathbb{R}^{1\times(J_i^L+5)}$

The $2(M - t_0)$ constraints can be written as:

$$\begin{bmatrix} v_{PG} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & v_{PG} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & v_{PG} \\ w_{PG} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & w_{PG} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & w_{PG} \end{bmatrix} x_i \leq 0_{2(M-t_0)\times 1}$$

Flexibility limitation of L

$$P_i^L(t) = \tilde{P}_i^L(t) \quad \rightarrow \quad \begin{cases} +\sigma_i^\top \delta_i^L(t) \leq +\tilde{P}_i^L(t) \\ -\sigma_i^\top \delta_i^L(t) \leq -\tilde{P}_i^L(t) \end{cases} \quad \forall t < t_i^{L,0} \vee t > t_i^{L,f}$$

Introducing vector:

$$v_{FL} = \begin{bmatrix} 0 & 0 & 0 & \sigma_i^\top & 0 & 0 \end{bmatrix}$$

Constraint (2.12) can be re-written as

$$\begin{bmatrix} v_{FL} & \zeta_{xi} & \cdots & \cdots & \cdots & \cdots & \zeta_{xi} \\ \zeta_{xi} & v_{FL} & \ddots & \ddots & \ddots & \ddots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & v_{FL} & \zeta_m & \cdots & \zeta_{xi} \end{bmatrix} x_i \leq \begin{bmatrix} \tilde{P}_i^L(t_0) \\ \tilde{P}_i^L(t_0+1) \\ \vdots \\ \tilde{P}_i^L(t_i^{L,0}) \end{bmatrix}$$

$$\begin{bmatrix} \zeta_{xi} & \cdots & \zeta_{xi} & v_{FL} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \cdots & \cdots & \cdots & \zeta_{xi} & v_{FL} \end{bmatrix} x_i \leq \begin{bmatrix} \tilde{P}_i^L(t_i^{L,f}+1) \\ \vdots \\ \tilde{P}_i^L(M-1) \end{bmatrix}$$

$$\begin{bmatrix} -v_{FL} & \zeta_{xi} & \cdots & \cdots & \cdots & \cdots & \zeta_{xi} \\ \zeta_{xi} & -v_{FL} & \ddots & \ddots & \ddots & \ddots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & -v_{FL} & \zeta_{xi} & \cdots & \zeta_{xi} \end{bmatrix} x_i \leq \begin{bmatrix} -\tilde{P}_i^L(t_0) \\ -\tilde{P}_i^L(t_0+1) \\ \vdots \\ -\tilde{P}_i^L(t_i^{L,0}) \end{bmatrix}$$

$$\begin{bmatrix} \zeta_{xi} & \cdots & \zeta_{xi} & -v_{FL} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \cdots & \cdots & \cdots & \zeta_{xi} & -v_{FL} \end{bmatrix} x_i \leq \begin{bmatrix} -\tilde{P}_i^L(t_i^{L,f}+1) \\ \vdots \\ -\tilde{P}_i^L(M-1) \end{bmatrix}$$

Cost Function Reformulation

$$\begin{cases} h_i^B(t) = \left\| P_i^B(t) - P_i^B(t-1) \right\| \\ h_i^L(t) = \left\| P_i^L(t) - \tilde{P}_i^L(t) \right\| \end{cases}$$

$$\rightarrow \begin{cases} +P_i^B(t) - P_i^B(t-1) - h_i^B(t) \leq 0 \\ -P_i^B(t) + P_i^B(t-1) - h_i^B(t) \leq 0 \\ +P_i^L(t) - \tilde{P}_i^L(t) - h_i^L(t) \leq 0 \\ -P_i^L(t) + \tilde{P}_i^L(t) - h_i^L(t) \leq 0 \end{cases} \quad \forall t = t_0, ..., M-1$$

Considered as known the value $P_i^B(t_0 - 1)$, exploiting vector v_{DCL} defined by (B) and:

- $v_{1,LCF} = \begin{bmatrix} 0 & 1 & 0 & 0_{1 \times J_i^L} & -1 & 0 \end{bmatrix}$
- $v_{2,LCF} = \begin{bmatrix} 0 & -1 & 0 & 0_{1 \times J_i^L} & -1 & 0 \end{bmatrix}$

it is possible to write the constraints (2.16a)-(2.16b) as:

$$\begin{bmatrix} v_{1,LCF} & \zeta_{xi} & \cdots & \cdots & \zeta_{xi} \\ -v_{DCL} & v_{1,LCF} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & -v_{DCL} & v_{1,LCF} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & -v_{DCL} & v_{1,LCF} \\ v_{2,LCF} & \zeta_{xi} & \cdots & \cdots & \zeta_{xi} \\ v_{DCL} & v_{2,LCF} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & v_{DCL} & v_{2,LCF} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & v_{DCL} & v_{2,LCF} \end{bmatrix} x_i \leq \begin{bmatrix} P_i^B(t_0 - 1) \\ 0 \\ \vdots \\ 0 \\ -P_i^B(t_0 - 1) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Whilst introducing

- $v_{3,LCF} = \begin{bmatrix} 0 & 0 & 0 & \sigma_i^\top & 0 & -1 \end{bmatrix}$
- $v_{4,LCF} = \begin{bmatrix} 0 & 0 & 0 & -\sigma_i^\top & 0 & -1 \end{bmatrix}$

$$\begin{bmatrix} v_{3,LCF} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & v_{3,LCF} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \ddots & \zeta_{xi} & v_{3,LCF} \\ v_{4,LCF} & \zeta_{xi} & \cdots & \zeta_{xi} \\ \zeta_{xi} & v_{4,LCF} & \cdots & \zeta_{xi} \\ \vdots & \ddots & \ddots & \vdots \\ \zeta_{xi} & \ddots & \zeta_{xi} & v_{4,LCF} \end{bmatrix} x_i \leq \begin{bmatrix} \tilde{P}_i^L(t_0) \\ \tilde{P}_i^L(t_0+1) \\ \vdots \\ \tilde{P}_i^L(M-1) \\ -\tilde{P}_i^L(t_0) \\ -\tilde{P}_i^L(t_0+1) \\ \vdots \\ -\tilde{P}_i^L(M-1) \end{bmatrix}$$

Correctness of the amount of energy supplied to L

$$\sum_{t=0}^{M-1} \Delta t_s P_i^L(t) = E_i^L \quad \rightarrow \quad \begin{cases} \sum_{t=0}^{M-1} \Delta t_s \sigma_i^\top \delta_i^L(t) \leq E_i^L \\ -\sum_{t=0}^{M-1} \Delta t_s \sigma_i^\top \delta_i^L(t) \leq -E_i^L \end{cases}$$

Defining:

$$\eta_{xi} = \begin{bmatrix} 0 & 0 & 0 & \sigma_i^\top & 0 & 0 \end{bmatrix}$$

Constraint (2.11) can be rewritten as

$$\begin{bmatrix} \eta_{xi} & \eta_{xi} & \cdots & \eta_{xi} \end{bmatrix} x_i \leq \frac{E_i^L}{\Delta t_s}$$
$$\begin{bmatrix} -\eta_{xi} & -\eta_{xi} & \cdots & -\eta_{xi} \end{bmatrix} x_i \leq -\frac{E_i^L}{\Delta t_s}$$

Rebound Effect Avoidance

$$P_i(t) = \tilde{P}_i(t)$$
$$\rightarrow \quad \begin{cases} +P_i^G(t) + P_i^B + \sigma_i^\top \delta_i^L(t) \leq \tilde{P}_i(t) \\ -P_i^G(t) - P_i^B - \sigma_i^\top \delta_i^L(t) \leq -\tilde{P}_i(t) \end{cases} \qquad \forall t = t_f + 1, ..., M-1$$

Defining vector

$$v_r = \begin{bmatrix} 1 & 1 & 0 & \sigma_i^\top & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \zeta_{xi} & \cdots & \zeta_{xi} & v_r & \zeta_{xi} & \cdots & \zeta_{xi} \\ \vdots & & & \ddots & & & \vdots \\ \zeta_{xi} & \cdots & \cdots & \cdots & \cdots & \zeta_{xi} & v_r \end{bmatrix} x_i \leq \begin{bmatrix} \tilde{P}_i(t_f+1) \\ \vdots \\ \tilde{P}_i(M-1) \end{bmatrix}$$

$$\begin{bmatrix} \zeta_{xi} & \cdots & \zeta_{xi} & -v_r & \zeta_{xi} & \cdots & \zeta_{xi} \\ \vdots & & & \ddots & & & \vdots \\ \zeta_{xi} & \cdots & \cdots & \cdots & \cdots & \zeta_{xi} & -v_r \end{bmatrix} x_i \leq \begin{bmatrix} -\tilde{P}_i(t_f+1) \\ \vdots \\ -\tilde{P}_i(M-1) \end{bmatrix}$$

The overall D_i matrix and d_i vector are then obtained by ordering all the matrices and vectors in column.

Coupling Constraints

The coupling constraint (2.13) concerns the overall power demand and hence cannot be reduced to a set of decoupled conditions on the prosumers. Nevertheless, it can be expressed in the form $\sum_{i=1}^{N} A_i \cdot x_i \leq b$:

$$\begin{cases} +P(t) - \tilde{P}(t) \leq +(1+\varepsilon)\Delta\ P(t) \\ -P(t) + \tilde{P}(t) \leq -(1-\varepsilon)\Delta\ P(t) \end{cases}$$

$$\rightarrow \begin{cases} +\sum_{i=1}^{m} P_i(t) \leq \tilde{P}(t) + (1+\varepsilon)\Delta_P(t) \\ -\sum_{i=1}^{m} P_i(t) \leq -\tilde{P}(t) - (1-\varepsilon)\Delta_P(t) \end{cases} \quad \forall t = t_0, ..., t_f$$

$$\begin{bmatrix} v_r & \zeta_{xi} & & \cdots & & & \zeta_{xi} \\ \vdots & & & \ddots & & & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & v_r & \zeta_{xi} & \cdots & \zeta_{xi} \end{bmatrix} x_i \leq \begin{bmatrix} \tilde{P}(t_0) + (1+\varepsilon)\Delta_P(t_0) \\ \vdots \\ \tilde{P}(t_f) + (1+\varepsilon)\Delta_P(t_f) \end{bmatrix}$$

$$\begin{bmatrix} -v_r & \zeta_{xi} & & \cdots & & & \zeta_{xi} \\ \vdots & & & \ddots & & & \vdots \\ \zeta_{xi} & \cdots & \zeta_{xi} & -v_r & \zeta_{xi} & \cdots & \zeta_{xi} \end{bmatrix} x_i \leq \begin{bmatrix} -\tilde{P}(t_0) - (1-\varepsilon)\Delta_P(t_0) \\ \vdots \\ -\tilde{P}(t_f) - (1-\varepsilon)\Delta_P(t_f) \end{bmatrix}$$

Bibliography

[1] Graham Goodwin, Mara M. Seron, and Jos A. de Don. *Constrained Control and Estimation: An Optimisation Approach*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[2] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics and constraints. *Automatica*, 35:407–427, 1999.

[3] M. Hejri and A. Giua. Hybrid modeling and control of switching DC-DC converters via MLD systems. In *2011 IEEE International Conference on Automation Science and Engineering*, pages 714–719, 2011.

[4] M. Mukai, T. Azuma, and M. Fujita. A collision avoidance control for multi-vehicle using pwa/mld hybrid system representation. In *Proceedings of the 2004 IEEE International Conference on Control Applications, 2004.*, volume 2, pages 872–877 Vol.2, Sep. 2004.

[5] Alessandra Parisio, Evangelos Rikos, and Luigi Glielmo. A model predictive control approach to microgrid operation optimization. *IEEE Transactions on Control Systems Technology*, 22(5):1813–1827, 2014.

[6] Alberto Bemporad, Domenico Mignone, and Manfred Morari. Moving horizon estimation for hybrid systems and fault detection. In *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, volume 4, pages 2471–2475. IEEE, 1999.

[7] A. Bemporad, F. Borrelli, and M. Morari. Piecewise linear optimal controllers for hybrid systems. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 2, pages 1190–1194 vol.2, 2000.

[8] Sertac Karaman, Ricardo G Sanfelice, and Emilio Frazzoli. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *2008 47th IEEE Conference on Decision and Control*, pages 2117–2122. IEEE, 2008.

[9] Ralph E Gomory. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem. In *50 Years of Integer Programming 1958-2008*, pages 77–103. Springer, 2010.

[10] Vasek Chvatal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973.

[11] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[12] Gérard Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, 2008.

[13] Ailsa H. Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[14] Robin Vujanic, Peyman Mohajerin Esfahani, Paul Goulart, Sébastien Mariéthoz, and Manfred Morari. A decomposition method for large scale milps, with performance guarantees and a power system application. *Automatica*, 67:144–156, 2016.

[15] Alessandro Falsone, Kostas Margellos, and Maria Prandini. A decentralized approach to multi-agent MILPs: finite-time feasibility and performance guarantees. *Automatica*, 103:141–150, 2019.

[16] Alessio La Bella, Alessandro Falsone, Ioli Daniele, Prandini Maria, and Scattolini Riccardo. A mixed-integer distributed approach to prosumers aggregation for providing balancing services. Submitted, 2020.

[17] Silvano Martello. Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimiza tion*, 1990.

[18] Philipp Baumann and Norbert Trautmann. Portfolio-optimization models for small investors. *Mathematical Methods of Operations Research*, 77(3):345–356, 2013.

[19] Daniel Pérez Palomar and Mung Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451, 2006.

[20] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[21] Arthur M Geoffrion. Lagrangean relaxation for integer programming. In *Approaches to integer programming*, pages 82–114. Springer, 1974.

[22] Ralph Gomory. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960.

[23] Alberto Caprara and Matteo Fischetti. Branch-and-cut algorithms. *Annotated bibliographies in combinatorial optimization*, pages 45–64, 1997.

[24] Lester Randolph Ford Jr and Delbert R Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[25] Dimitri Bertsekas, G Lauer, N Sandell, and T Posbergh. Optimal short-term scheduling of large-scale power systems. *IEEE Transactions on Automatic Control*, 28(1):1–11, 1983.

[26] N Jiménez Redondo and AJ Conejo. Short-term hydro-thermal coordination by lagrangian relaxation: solution of the dual problem. *IEEE transactions on power systems*, 14(1):89–95, 1999.

[27] Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli, et al. *Integer programming*, volume 271. Springer, 2014.

[28] Alessandro Falsone, Kostas Margellos, Simone Garatti, and Maria Prandini. Dual decomposition for multi-agent distributed optimization with coupling constraints. *Automatica*, 84:149–158, 2017.

[29] Cevdet Aykanat, Ali Pinar, and Umit Catalyurek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on Scientific Computing*, 25, 12 2002.

[30] Martin Bergner, Alberto Caprara, Alberto Ceselli, Fabio Furini, Marco E Lübbecke, Enrico Malaguti, and Emiliano Traversi. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1-2):391–424, 2015.

[31] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[32] George Karpys, Rajat Aggarwal, Vipin Kumar, and Sashi Shekar. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, 1:69–78, 03 1999.

[33] Charles J Alpert, Jen-Hsin Huang, and Andrew B Kahng. Multilevel circuit partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):655–667, 1998.

[34] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. K-way hypergraph partitioning via n-level recursive bisection. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 53–67. SIAM, 2016.

[35] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th design automation conference*, pages 175–181. IEEE, 1982.

[36] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[37] Daniel Schweikert and B. Kernighan. A proper model for the partitioning of electrical circuits. *Proc. 9th Design Automation Workshop*, pages 57–62, 01 1972.

[38] Thang Bui, Christopher Heigham, Curt Jones, and Tom Leighton. Improving the performance of the kernighan-lin and simulated annealing graph bisection algorithms. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 775–778, 1989.

[39] Charles J Alpert and Andrew B Kahng. A general framework for vertex orderings with applications to circuit clustering. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(2):240–246, 1996.

[40] Thang Nguyen Bui and Curt Jones. A heuristic for reducing fill-in in sparse matrix factorization. Technical report, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA ..., 1993.