UCLouvain
Institute for Information
and Communication Technologies,
Electronics and Applied Mathematics

Computer-Aided Analysis of Decentralized Optimization Methods

Sébastien Colla

Thesis submitted in partial fulfillment
of the requirements for the degree of
Docteur en sciences de l'ingénieur

Dissertation committee:

Prof. Julien M. Hendrickx (UCLouvain, advisor)
Prof. François Glineur (UCLouvain)
Prof. Alex Olshevsky (Boston University, USA)
Prof. Angelia Nedich (Arizona State University, USA)
Prof. Bryan Van Scoy (Miami University, USA)
Prof. Pierre-Antoine Absil (UCLouvain, chair)

Version of August 29, 2024.

Abstract

This thesis develops and exploits computational techniques for automatically and accurately calculating the worst-case performance of decentralized optimization algorithms. These algorithms aim to minimize the average of local functions that are distributed across a network of computers (called agents), with many applications in machine learning, sensor networks, and power systems. Decentralized optimization algorithms are iterative, with agents alternating between local computations and communication steps. Despite the abundance of decentralized algorithms, accurately analyzing their worst-case performance remains challenging. This analysis is crucial for their effective application, parameter tuning, and comparison. Therefore, we propose a computer-aided methodology based on the Performance Estimation Problem (PEP) approach, which computes the worst-case performance and identifies a worst-case instance of first-order optimization algorithms by solving a semidefinite program.

In the first part of the thesis, we extend the PEP framework to decentralized optimization. For this purpose, we propose a practical description of the effects of a class of communication steps in the algorithm. The development of decentralized PEP formulations also allows us to identify conditions that ensure the independence of the worst-case performance from the total number of agents. These conditions are met by many algorithms in various settings, enabling the analysis of their performance in the basic case with just two agents.

In the second part, we use our new PEP-based techniques to analyze, tune, and compare several decentralized optimization algorithms. Our results improve on the literature, address open questions, and offer new analyses and comparisons. Additionally, we investigate the optimal communication weights to be used in decentralized optimization given a specific network topology. Our analysis reveals that current weight selection strategies are not optimal, leaving room for performance improvement.

Remerciements

Ces quatre années de doctorat furent pour moi comme un voyage. Un voyage parfois sinueux, parfois challengeant et parfois merveilleux ! Il m'a permis de rencontrer et de collaborer avec des personnes exceptionnelles, riches en compétences et en conseils, qui m'ont motivé et fait progresser. Car, plus encore que la destination ou le chemin, ce sont les compagnons de routes qui ont le plus compté pour moi. Je tiens ici à toutes et tous les remercier comme il se doit.

Tout d'abord, merci à mon superviseur Julien Hendrickx, pour ses conseils avisés, son soutien, son écoute et sa motivation. Il a su me donner confiance en moi et m'a poussé à produire de la recherche de qualité.

Merci également aux membres de mon comité d'accompagnement : François Glineur et Alex Olshevsky, ainsi qu'aux autres membres de mon jury de thèse : Angelia Nedich, Bryan Van Scoy et Pierre-Antoine Absil. Ils et elle m'ont accordé un temps précieux pour enrichir mon travail via les discussions, conseils et remarques encourageantes. En particulier, j'aimerais remercier François pour m'avoir donné le goût de l'optimisation durant les cours de bachelier et de master. Un grand merci aussi à Alex pour son accueil à Boston, c'était un honneur pour moi de collaborer avec lui. Enfin, merci à Bryan pour son accueil amical dans l'Ohio, j'y ai passé de très beaux moments, tout en profitant de son expertise et de son point de vue complémentaire sur mon domaine de recherche.

Je tiens également à remercier Adrien Taylor pour son aide précieuse et bienveillante, notamment concernant les codes, la toolbox et la théorie PEP en général. Merci à Balazs Gerencser pour ses visites annuelles, toujours riches d'idées et de discussions. Au jour le jour, j'ai également eu la chance d'échanger et de collaborer avec d'autres formidables doctorant·es en optimisation : Anne, Nizar et Yassine, alias la team PEP, pour le meilleur et pour le pire.

Par ailleurs, je suis reconnaissant envers toutes les chouettes personnes rencontrées en conférence : David, qui m'a accompagné à quasiment chacune de mes conférences, Baptiste et Céline, dont la rencontre en conférence nous a mené à organiser le workshop « PEP-it » et Shuvo.

Au-delà du travail scientifique, je tiens à remercier toutes les personnes qui ont contribué à faire de l'Euler un cadre de travail si agréable et épanouissant et qui m'ont permis de vivre mon périple plus sereinement. Tout d'abord merci à mes collègues de bureau : Mingming, Isabella, et surtout Renato, ce roi des seconds prénoms, à l'accent espagnol exquis, au déhanché extraordinaire et avec qui c'était si chouette de partager mon bureau à la cave. Ses relectures et ses conseils furent précieux. Merci aussi à toute la team JUL pour l'ambiance de groupe, les délires, l'entre-aide et les lunchs à la branche du figuier. En particulier, merci à Charles pour ses perles de sagesse, ses conseils rassurants et ses références obscures ; merci à Tonio pour sa bonne humeur et les discussions philo ; merci à Yassine pour la PunchNight et les tournois de spikeball, merci à Nizar pour son courage capillaire et son talent aux jeux de société et merci à Anne pour d'innombrables raisons, comprenant ses relectures, les pauses dans son bureau, les voyages à vélo en conférence et nos longues discussions rigolotes et bienveillantes. Merci à Renato, Moslem, Martina et Erwan d'être venu·es compléter cette incroyable équipe.

Merci à la team des chevaliers Euler qui a vécu plus d'une Ribouldingue et d'autres riches aventures. En particulier, merci à Clem pour son humour et son amitié, Phil pour sa folie et sa motivation à toute épreuve et Julien, alias le concierge de l'Euler, pour sa gentillesse et sa détermination. Plus généralement, merci à toutes les autres belles personnes que j'ai eu la chance de côtoyer à l'Euler et qui m'ont donné envie de venir jour après jour, pour profiter des pauses tartes, spikeball, potins et autres : Michel, Adrien, Astrid, Olivier, Matteo, Lucas, Virginie, Rémi, Antoine, Simon, Téo, Brieuc, et tous les autres.

Merci à tous mes collègues de course, en particulier les Pioupious, qui m'ont encouragé à faire du sport régulièrement, ce qui est très important dans l'équilibre d'un doctorant. Merci aussi à Vincent Legat pour les chouettes discussions et sa bonne humeur contagieuse.

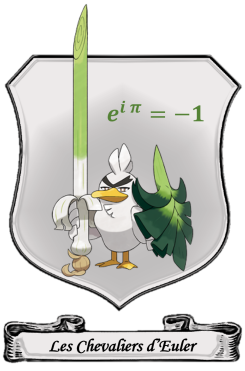
Merci aux anciens Euleriens qui m'ont bien accueilli lors de mon arrivé, malgré le covid : Charles, Loic, Cécile, Guillaume, Guillaume et Emilie.

Un grand merci à Marie-Christine, Pascale, Etienne, François, Astrid, Arnaud, John et Marine qui ont rendu ma vie de chercheur à l'Euler plus facile et bien plus belle !

Je tiens aussi à remercier sincèrement tous les membres d'Ingés en Transition. Merci pour toutes nos discussions, l'énergie déployée et l'impact recherché. Je suis heureux et fier d'avoir fait partie de ce collectif. Qu'est-ce que c'est encourageant de lutter ensemble pour des valeurs partagées, dans un esprit amical et joyeux.

En dehors du cadre de travail, j'ai aussi eu la chance d'être superbement bien soutenu par mes proches, que je remercie chaudement. Merci à la coloc des pommiers, Cam, Vic, Léo, Ed et Sandrine, pour tous ces moments de vie, de rigolade, de questionnements et de camaraderie. Merci à mes parents, mon frère et ma sœur pour leur soutien inconditionnel dans mes projets, en l'occurrence ici, une thèse de doctorat. Et surtout, merci à Alexia de m'avoir supporté, épaulé, écouté, conseillé et motivé tout au long de mon périple.

Mille merci pour,
Vos sourires et vos conseils,
Tout au long du chemin.



Contents

Contents . . . . . **v**

List of symbols . . . . . **ix**

List of acronyms . . . . . **xii**

1 Introduction **1**
1.1 Outline and contributions 5

2 Decentralized Optimization **8**
2.1 Decentralized consensus optimization problem 9
2.2 Relevance and applications 12
2.3 Communications and average consensus 16
2.3.1 Setting considered: synchronous and undirected communications 17
2.3.2 Consensus convergence results 18
2.3.3 Related work 21
2.4 Existing algorithms for decentralized optimization 23
2.5 The class of decentralized optimization methods considered . . 33
2.6 Performance evaluation of decentralized methods 35
2.7 Vector notations 36

3 Performance Estimation Problems **38**
3.1 Elements of a PEP 39
3.2 Finite formulation and convex interpolation 40
3.3 SDP reformulation 42
3.4 Exploitation of PEP 44
3.5 An alternative approach from control theory 47

I Performance Estimation Problems for Decentralized Optimization

Visual outline of Part I **51**

4 The agent-dependent PEP formulations **52**
4.1 High-level PEP description 53
4.2 SDP reformulation and Gram-representable settings 54
4.3 Spectral representation of consensus steps in PEP 60
4.3.1 The intractable quest for tightness 61
4.3.2 A relaxed formulation 63
4.3.3 Tightness check and worst-case averaging matrix 70
4.3.4 Impact of the relaxation 74
4.4 Example: the exact and spectral PEP formulations for DGD . 77
4.5 Numerical results 79
4.5.1 Independence of the number of agents 81
4.5.2 Impact of relaxations 81
4.6 Summary and conclusion 84

5 Agent-independent relaxed PEP formulation **86**
5.1 PEP with global vector and function representation 87
5.2 Relax the separability of the objective function 88
5.3 Change of variables and new coordinates system 90
5.4 Numerical comparison with the agent-dependent PEP 99

6 Exploiting agents symmetries in PEP **102**
6.1 Equivalence classes of agents in PEP 104
6.2 Agent-independent PEP when all agents are equivalent 109
6.2.1 The PEP formulation restricted to fully symmetric solutions . 110
6.2.2 Impact of agent symmetry on the worst-case solution 118
6.3 Compact PEP with multiple equivalence classes of agents . . . 119
6.3.1 The PEP formulation restricted to symmetric agent-class solutions 121
6.3.2 Infinitely many agents 124
6.4 Comparison with the agent-dependent PEP formulation 125
6.5 Summary and conclusion 126

Appendix of Chapter 6 . . . . . **127**
6.A Explicit expressions of the compact PEP components 127
6.A.1 When all agents are equivalent 127

6.A.2 With multiple equivalence classes of agents . . . 138

7 How to use in practice? . . . **145**
7.1 Implementation . . . 145
7.2 Which formulation to choose? . . . 148

II Exploitation for analysis and tuning

8 Algorithms analysis . . . **153**
8.1 Distributed (sub)gradient descent (DGD) . . . 154
8.1.1 Comparison with Theorem 8.1 . . . 156
8.1.2 Worst-case averaging matrix and tightness analysis . . . 157
8.1.3 Evolution with the total number of iterations t . . . 157
8.1.4 Tuning the step-size α . . . 157
8.1.5 Note on scaling of DGD . . . 159
8.2 DIGing . . . 160
8.2.1 Impact of the number of agents n . . . 163
8.2.2 Comparison between the theoretical and the spectral PEP bounds 163
8.2.3 Convergence rate analysis with PEP . . . 165
8.2.4 Impact of the step-size α . . . 167
8.2.5 Impact of time-varying averaging matrices . . . 169
8.2.6 Worst-case averaging matrix and tightness analysis . . . 169
8.3 Accelerated Distributed Nesterov Gradient Descent . . . 170
8.3.1 Theoretical performance guarantee from [QL19] . . . 171
8.3.2 Performance analysis with PEP . . . 171
8.3.3 Recent theoretical results from [LL21] . . . 175
8.4 EXTRA . . . 176
8.4.1 Performance of the worst agent . . . 177
8.4.2 The 80-th percentile of the agent performance . . . 180
8.4.3 Performance under local functions heterogeneity . . . 183
8.5 Comparison of algorithms . . . 186
8.5.1 Analysis of the results . . . 189
8.5.2 Adapt-then-Combine (ATC) variants . . . 191

9 On the Optimal Communication Weights . . . **194**
9.1 Weights Heuristics . . . 196
9.1.1 Conventional choices of weights from average consensus . . . 196
9.1.2 Other weight heuristics based on the full spectrum . . . 198

9.2 Worst-case Performance evaluation 200
9.3 Weights minimizing the worst-case performance 201
9.4 Results 202
9.4.1 Weights minimizing λ (SLEM) are not optimal 204
9.4.2 Weights heuristics analysis 205
9.4.3 Optimal weights 208
9.4.4 Results for two random networks 208
9.5 Summary and conclusion 209

III Conclusion

10 Research outcomes and perspective 211
10.1 Research outcomes 212
10.2 Perspectives and open questions 215

List of Publications 220

Bibliography 222

List of symbols

This section lists the main notations introduced throughout this thesis with a reference to the page where they are defined or appear for the first time. They are listed in alphabetical order, starting with the typographical symbols, then the Latin symbols, and finally Greek symbols.

$|\cdot|$ — absolute value (p. 9)

$\|\cdot\|$ (or $\|\cdot\|_2$) — Euclidean norm (p. 9)

$\|\cdot\|_F$ — Frobenius norm (p. 71)

$\|\cdot\|_\Sigma$ — nuclear norm of a matrix (p. 196)

$\|\cdot\|_{\Sigma,\gamma}$ — weighted nuclear *norm* of a matrix, with coefficients γ (p. 196)

$\mathbf{1}$ — vector of ones (p. 19)

$\succeq$ — generalized inequality symbol for the positive semidefinite cone (p. 42)

$\otimes$ — Kronecker product (p. 36)

A — (decentralized) algorithm to analyze (p. 35)

$\mathcal{A}_D$ — class of decentralized optimization algorithms considered (p. 34)

$\mathcal{C}$ — consensus space (p. 11)

$\mathcal{C}^\perp$ — orthogonal complement of the consensus space (p. 11)

d — dimension of the function to be optimized (p. 9)

$\mathcal{E}$ — set of edges in the communication graph $\mathcal{G}$ (p. 8)

e_x — vector of coefficients such that $P_i e_x = x_i$ (p. 103)

f — global objective function (p. 9)

F^{s_n} — a separable objective function (p. 11)

$\mathbf{f}$ — vector of function values (p. 42)

f_i — local function of agent i (p. 9)

$\mathcal{F}$ — class of (local) functions (p. 35)

$\mathcal{F}_{\mu,L}$ — set of all L-smooth and μ-strongly convex functions (p. 40)

$\mathcal{F}_R$ — set of all R-Lipschitz convex functions (p. 40)

$\mathcal{G}$ — graph over which the agents can interact (p. 8)

G — Gram matrix (p. 42)

G_{ij} — block of the Gram matrix G with scalar products between variables of agent i and those of agent j (p. 103)

$\mathcal{I}$ — set of initial conditions (p. 35)

L — smoothness parameter (p. 10)

m — number of edges in the communication graph $\mathcal{G}$ (p. 8)

$\mathcal{M}_{(\lambda^-,\lambda^+)}$ — set of symmetric linear operators preserving the consensus subspace, with a given range of eigenvalues $[\lambda^-, \lambda^+]$ (p. 64)

n — number of agents in the system (p. 8)

n_u — number of agents in the subset $\mathcal{V}_u$ (p. 104)

P — matrix factorizing the Gram matrix, i.e., $G = P^T P$ (p. 42)

P_i — matrix factorizing the block G_{ii} of the Gram matrix G (p. 103)

$\mathcal{P}$ — performance criterion to evaluate (p. 35)

r — number of agent equivalence classes in $\mathcal{V}$ (p. 104)

R — Lipschitz constant (p. 9)

$\mathbb{R}$ — real numbers (p. 2)

$\mathcal{S}_n$ — performance evaluation settings (p. 35)

t — total number of iterations (p. 35)

$\mathcal{T}$ — partition of the set of agents $\mathcal{V}$ (p. 104)

$\mathcal{V}$ — set of agents in the system (p. 8)

$\mathcal{V}_u$ — subset of agent composing the equivalence class u (p. 104)

W — averaging matrix (p. 18)

w_{ij} — communication weight between agent i and j (p. 16)

$\mathcal{W}$ — class of averaging matrices (p. 35)

$\mathcal{W}_{(\lambda^-,\lambda^+)}$ — class of symmetric and generalized doubly stochastic matrices with range of eigenvalues given by $[\lambda^-, \lambda^+]$ (p. 60)

$w(\mathcal{S}_n)$ — worst-case performance associated with settings $\mathcal{S}_n$ (p. 55)

y_i^k — local variable of agent i resulting from consensus step k (p. 33)

x_i^k — local variable x^k held by agent i, associated with iteration k (p. 36)

$\mathbf{x}^k$ — vector stacking the local variables of each agent $x_1^k, \dots, x_n^k$ (p. 36)

$\overline{x}^k$ — agent average of vector $\mathbf{x}^k$ (p. 37)

x_{av} — running average of agent average $\overline{x}^k$ (p. 37)

X — matrix with $\mathbf{x}^0, \dots, \mathbf{x}^t$ as columns (p. 36)

$\mathbf{y}^k$ — vector stacking the local variables of each agent $y_1^k, \dots, y_n^k$ (p. 36)

Y — matrix with $\mathbf{y}^0, \dots, \mathbf{y}^t$ as columns (p. 36)

α — step-size used in the algorithm (p. 24)

$\lambda(W)$ or (λ) — second-largest eigenvalue modulus of matrix W (p. 20)

$\lambda_i(W)$ (or λ_i) — the i-th largest eigenvalue of matrix W (p. 19)

μ — strong convexity parameter (p. 10)

List of acronyms

Acc-DNGD	**Acc**elerated **D**istributed **N**esterov **G**radient **D**escent
ATC	**A**dapt **T**hen **C**ombine
CTA	**C**ombine **T**hen **A**dapt
DGD	**D**istributed **G**radient **D**escent
DIGing	**D**istributed **I**nexact **G**radient method with gradient track**ing**
EXTRA	**EX**act firs**T**-orde**R** **A**lgorithm
IQC	**I**ntegral **Q**uadratic **C**onstraints
PEP	**P**erformance **E**stimation **P**roblem
PESTO	**P**erformance **E**stimation **TO**olbox
SDP	**S**emi-**D**efinite **P**rogramming
SLEM	**S**econd-**L**argest **E**igenvalue **M**odulus

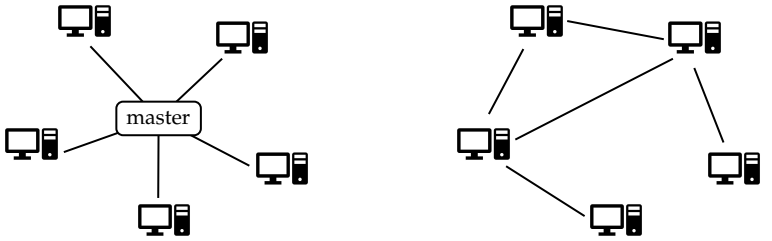
L'union fait la force.

Belgian motto

1 Introduction

WHILE we can achieve great things alone, our potential multiplies when we work together. What if the same applied to computers? This is the matter of this thesis: how to make computers cooperate to solve a problem. And by computer, we mean any system or object with computation and communication capabilities: a smartphone, a laptop, a server, a sensor device, a robot, etc. Such entities are called *agents*. The recent advances in digital systems and communication technologies have led to the development of networked systems and multi-agent systems, which consist of interconnected agents that cooperate to achieve a desirable global goal. For example, in sensor networks, the goal of the sensors could be to estimate a parameter from the collected data; in machine learning, the goal of the servers could be to train a model with a data set spread across different machines, for privacy or storage issues; in multi-robot systems, the goal of the robots could be to agree on a rendezvous that minimizes the total energy expended by the robots. Depending on the application, the agents cooperate through a centralized or decentralized network architecture, see Fig. 1.1. Unlike the decentralized system, the centralized one has a central authority (master node) that coordinates the agents to solve the problem. In many applications, the decentralized architecture is preferred for several reasons:

- **Scalability**: a centralized architecture requires the master node to have many more resources to work over large systems, i.e., with many agents. The central unit may become a communication bottle-



(a) Centralized **(b)** Decentralized

Fig. 1.1 Examples of centralized and decentralized network architectures

neck because it needs to communicate with every single other agent.

- **Robustness**: a centralized network architecture is not robust to the failure of the central node. Hence, the central unit may be seen as a robustness bottleneck.
- **Privacy**: in general, the privacy of agent information cannot be guaranteed in centralized systems.

Many goals in multi-agent systems can be posed as a decentralized consensus optimization problem, where the n agents try to minimize a global objective function f, which is the sum or the average of local functions $f_i : \mathbb{R}^d \to \mathbb{R}$ $(i = 1, \ldots, n)$, privately held by each agent i:

$$\min_{x \in \mathbb{R}^d} \quad f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{1.1}$$

The agents aim to solve such a problem in a distributed manner, using local computations and communications. The decentralized consensus optimization problem is presented in detail in Chapter 2. It has long been studied, with seminal work dating back to [Ber83, Tsi84, TBA86] in the context of parallel and distributed computation. Over the past decade, this problem has received renewed interest, due to its diverse applications in machine learning, sensor networks, distributed control, and power systems, see e.g., [RN04, FCG10, NL18, CHW+20, HSYS21] and Section 2.2.

As detailed in Section 2.4, various iterative methods have been developed to solve Problem (1.1) in a distributed manner. In such methods, each agent i starts with a different estimate x_i of the solution and updates it to gradually agree on a common point x^*, which minimizes the global

objective function f. The first and simplest algorithm is probably the Distributed Gradient Descent (DGD) [NO09], where agents update their estimate by combining an average consensus step (1.2) and a local gradient step (1.3):

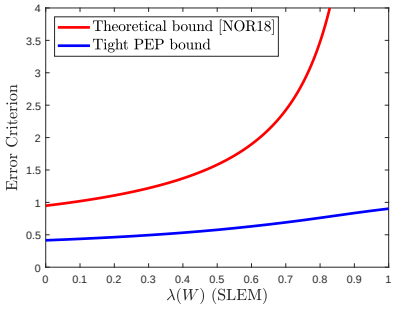
$$y_i^k = \sum_{j=1}^{n} w_{ij}^k x_j^k, \qquad \text{for } i = 1, \dots, n, \tag{1.2}$$

$$x_i^{k+1} = y_i^k - \alpha^k \nabla f_i(x_i^k), \qquad \text{for } i = 1, \dots, n, \tag{1.3}$$

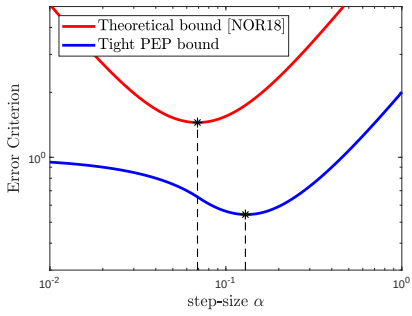
where $\alpha^k > 0$ are the step-sizes and w_{ij}^k are the mixing weights used in the consensus. While DGD is simple and intuitive, it does not feature very good convergence properties, as it will be detailed in Section 2.4. Many other algorithms have thus been proposed, generally consisting of specific linear combinations between local computations, involving gradient computations, and local communications, involving average consensus steps via the dedicated communication network. These include a wide variety of methods, which can have better convergence properties and be adapted to different problem settings (type of functions, type of network, etc.).

In this thesis, we focus on undirected synchronous communications and on the case where the mixing matrices $W^k = [w_{ij}^k]$ used in the consensus steps are symmetric, with rows and columns summing to one, and matching the topology of the communication network, in which case we call it *averaging matrices*. These assumptions are standard to ensure the convergence of many decentralized algorithms. Precise definitions and assumptions will be presented in Chapter 2.

As with classical optimization, the performance (i.e., the convergence speed) of decentralized optimization methods is generally assessed on the basis of worst-case guarantees. In practice, one analyzes what is the worst-case error produced by the algorithm after a given number of iterations t, and how this error evolves when t increases or the algorithm parameters, such as the step-size, change. Different error criteria can be analyzed, e.g., error on the agents iterates, the functions values, or the global gradient norm. Accurate guarantees on the worst-case performance of decentralized algorithms are essential to fully understand how their performance is influenced by their parameters, the conditioning of the objective functions, and the network topology, which then allows the different algorithms to be tuned and compared correctly. However, deriving such bounds can often be a challenging task, requiring a careful combination of the impacts of the optimization and communication components, which may result in



(a) Evolution with $\lambda(W)$, $\alpha = \frac{1}{\sqrt{t}}$.



(b) Evolution with α, $\lambda(W) = 0.8$.

Fig. 1.2 Comparison of the theoretical worst-case performance bounds for DGD [NOR18] with the tight bounds derived in this thesis. The bounds apply after $t = 10$ iterations of DGD with a time-invariant step-size α and a time-invariant averaging matrix $W = [w_{ij}]$ that has a given bound on its second-largest eigenvalue modulus (SLEM) $\lambda(W)$ (which will be formally defined in Chapter 2, see (2.8)).

conservative or highly complex bounds. Moreover, theoretical analyses often require specific settings that differ from one algorithm to another, making them difficult to compare. For example, for the simple DGD algorithm, Fig. 1.2 shows that the existing bounds [NOR18] for a time-invariant step-size α is rather conservative, especially for large second-largest eigenvalue modulus $\lambda(W)$ (which corresponds to poorly connected networks), or small step-sizes. Moreover, Fig. 1.2b shows how this non-tight bound leads to a suboptimal choice for the step-size tuning in DGD.

This thesis helps to solve such issues by developing tools allowing to automatically and accurately compute numerical guarantees on the worst-case performance of decentralized optimization algorithms. Our new tools rely on a computational approach that finds a worst-case performance guarantee of an algorithm by solving an optimization problem, known as the performance estimation problem (PEP), recently introduced in [DT12, Tay17] and which is detailed in Chapter 3. Our new tools enable computer-aided analysis of many decentralized algorithms, which can lead to improvements in their performance guarantees and parameter tuning, and allow rapid exploration of new algorithms. The development of the tools is the focus of Part I of the thesis and their exploitation is the focus of Part II. The next section overviews the content and contributions of the thesis in more details.

1.1 Outline and contributions

This thesis develops and exploits tools that automatically compute accurate numerical guarantees on the worst-case performance of decentralized optimization algorithms. In addition, the development of such techniques reveals fundamental properties of worst-case performance in decentralized optimization, such as conditions of independence of guarantees in terms of the number of agents and the agent symmetry of the worst-case local functions and variables. We detail the thesis organization and our contributions below.

Preliminaries

- **Chapter 2** details the decentralized consensus optimization problem (1.1) that we are considering in this thesis. It presents some examples of applications and reviews existing algorithms to solve it. In this chapter, we also define the type of agent interactions and the class of decentralized methods that we will focus on in this thesis.
- **Chapter 3** presents the Performance Estimation Problem (PEP) for classical first-order optimization methods [THG15b, THG15a]: its principle, its formulation, and its resolution. PEP can compute a tight worst-case performance guarantee of an optimization method by solving an optimization problem. Understanding the general PEP theory allows us to extend it to decentralized optimization in Part I of the thesis.

Part I: Performance Estimation Problems for Decentralized Optimization

In this part, we develop different PEP formulations for decentralized optimization algorithms. The specificities of each formulation, as well as their relations are summarized in a diagram at the beginning of Part I.

- In **Chapter 4**, we develop an *exact* and a *spectral agent-dependent PEP formulations* which both search for the worst-case local functions and iterates of each agent. The first formulation gives exact worst-case performance but is specific to a given averaging matrix. The second formulation provides worst-case guarantees valid over an entire spectral class of averaging matrices, characterized by the range of their eigenvalues. To build this spectral formulation, we propose a practical representation of consensus steps in PEPs. The tightness

of the results is not a priori guaranteed but can be verified a posteriori, by recovering a valid worst-case averaging matrix. While the resulting PEP size depends on the number of agents in the system, in many cases, we observe that the result of the spectral formulation is independent of the number of agents, which motivates the formulations of the next chapters. This work combines results from [CH23] and [CH24a].

- **Chapter 5** proposes a *relaxed agent-independent PEP formulation* with a problem size independent of the number of agents. It takes a global system view on the problem and introduces a change of variables. While the formulation presents another degree of relaxation in comparison with the agent-dependent one, it recovers the same results in all our experiments. This work is published in [CH22].
- **Chapter 6** exploits symmetries between agents in the spectral agent-dependent PEP formulation to obtain an equivalent *compact PEP formulation*, whose size depends only on the number of equivalence classes of agents, and not on the total number of agents. We also characterize the situations where the worst-case performance of a decentralized algorithm is totally independent of the number of agents, which allows analyzing and computing the performance in the fundamental case with only two agents. Such situations include many common settings for the performance evaluation of decentralized optimization algorithms, and require, in particular, that all the agents play an equivalent role in the algorithm and its performance evaluation. Finally, we also show that in the worst-case scenario, when all agents are equivalent, their local sequences of iterates are rotations of each other and their local functions are identical up to a change of variables. This work is contained in [CH24a].
- **Chapter 7** provides some details about the implementation of each formulation and explains which one to use depending on the algorithm and performance settings.

Part II: Exploitation for analysis and tuning

The PEP formulations from Part I allow analyzing the worst-case performance of any decentralized optimization algorithm that linearly combines local gradient samples with average consensus steps. In this second part, we exploit these formulations to analyze, compare and tune different decentralized optimization methods.

- **Chapter 8** analyzes the worst-case performance of several algorithms that are significant in the literature, namely, DGD [NO09], DIGing [NOS16], Acc-DNGD [QL19] and EXTRA [SLWY14]. Our PEP analysis significantly improves on the theoretical bounds from the literature, allows for better tuning of the step-size in the algorithms, and reveals that many algorithms have the same simple form for their worst-case averaging matrix. For Acc-DNGD, our PEP tool allows us to test the convergence conjectures stated in [QL19]. For EXTRA, we explore relevant and previously unstudied settings, including the performance of the worst agent, the 80-th percentile of the agent performance, and the average performance under heterogeneous conditioning of local functions. Unlike classical settings, these worst-case performances do depend on the number of agents n. Using the compact PEP formulation, we examine their evolution with n, including when $n \to \infty$. Finally, the chapter also compares these algorithms, and a few others, in equivalent and fair settings, revealing in particular that ATC-DIGing outperforms DIGing and that EXTRA is not robust to time-varying averaging matrices. This work combines results from [CH23], [CH22] and [CH21].
- **Chapter 9** leverages the exact agent-dependent PEP formulation to identify the best communication weights to use in decentralized optimization algorithms when the network topology is given. We establish that in decentralized optimization, the prevalent strategy of minimizing the second-largest eigenvalue modulus (SLEM) of the averaging matrix for selecting communication weights, while optimal for existing theoretical performance bounds, is generally not optimal regarding the exact worst-case performance of the algorithms. This suggests that the SLEM is not the best characterization of weighted network performance for decentralized optimization. Additionally, we explore and compare alternative heuristics for weight selection. This work is published in [CH24b].

Part III: Conclusion

We conclude the thesis by summarizing the research outcomes and discussing the perspectives in **Chapter 10**.

Great things are not done by impulse, but by a series of small things brought together.

Vincent Van Gogh

2

Decentralized Optimization

In decentralized optimization, one considers a set of agents $\mathcal{V} = \{1, \dots, n\}$, connected by a set $\mathcal{E}$ of m communication links via the network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, see Fig. 2.1. Agents may try to achieve various goals together, many of which can be formulated as a decentralized consensus optimization problem. In this chapter, we define this problem, provide some examples of applications, and describe existing algorithms to solve it. We also define the type of agent interactions and the class of decentralized methods on which we will focus in this thesis.

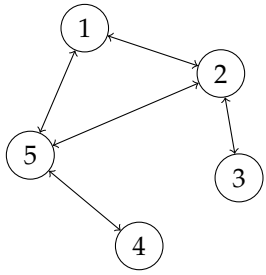


Fig. 2.1 An example of communication network $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}| = 5$ agents (nodes) and $m = |\mathcal{E}| = 5$ communication links (edges).

2.1 Decentralized consensus optimization problem

The agents work together, in a decentralized way, to minimize the average (or the sum) of their local functions $f_i : \mathbb{R}^d \to \mathbb{R}$,

$$\min_{x \in \mathbb{R}^d} \quad f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{2.1}$$

Each agent i can perform local computation on its own local function f_i and exchange local information with its neighbors in $\mathcal{G}$ to update its own estimate of the solution, denoted $x_i \in \mathbb{R}^d$. The agents seek to reach agreement on a minimizer x^* of the global function f, i.e., $x_1 = \cdots = x_n = x^*$. This thesis focuses on the cases where the local functions f_i are real-valued and convex (and so is the global function f) and where the global function f has a non-empty set of minimizers $\mathcal{X}^*$, so that the problem is well-defined.

Definition 2.1. (Convex function) A real function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if, for all $\gamma \in [0, 1]$ and all $x, y \in \mathbb{R}^d$, it holds

$$f(\gamma x + (1 - \gamma)y) \leq \gamma f(x) + (1 - \gamma) f(y).$$

There exist many characterizations of convex functions, see e.g., [Nes18]. Convexity is widely used in optimization theory because it ensures that any local minimum is also a global minimum, allowing one to rely on local information to minimize the function. In this thesis, we will sometimes work with additional assumptions about the objective functions.

Definition 2.2 (Lipschitz continuous)**.** A function $f : \mathbb{R}^d \to \mathbb{R}$ is Lipschitz continuous with constant $R \geq 0$, also called R-Lipschitz, if

$$|f(x) - f(y)| \leq R\|x - y\|, \qquad \text{for all } y, x \in \mathbb{R}^d.$$

When f is convex, then it is R-Lipschitz if all its subgradients are bounded,

$$\|g\| \leq R \quad \forall g \in \bigcup_{x \in \mathbb{R}^d} \partial f(x),$$

where $\partial f(x)$ denotes the subdifferential of function f at point x.

Definition 2.3. (Smooth) A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if its gradient is Lipschitz continuous, with constant $L \geq 0$,

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \qquad \text{for all } y, x \in \mathbb{R}^d.$$

When the function is convex, the smoothness condition is equivalent requiring the function to be upper-bounded as

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|y - x\|^2, \qquad \text{for all } y, x \in \mathbb{R}^d. \quad (2.2)$$

Definition 2.4. (Strongly convex) A differentiable convex function $f : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly convex, with $\mu \geq 0$, if the function is lower bounded as follows

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2}\|y - x\|^2 \qquad \text{for all } y, x \in \mathbb{R}^d. \quad (2.3)$$

We generally impose convex local functions to be Lipschitz continuous to ensure convergence of the algorithms in the worst-case, even when they are not differentiable everywhere. On the other hand, when the local functions are smooth and strongly convex, this enables well-designed algorithms to converge geometrically to a minimizer $x^* \in \mathcal{X}^*$ of the global function f in Problem (2.1). Such a minimizer $x^* \in \mathcal{X}^*$, which agents try to approximate collectively, satisfies the system-level stationarity condition:

$$\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^*) = 0, \qquad \text{for differentiable functions,}$$

where ∇f_i denotes the gradient of f_i. This condition also extends to non-differentiable convex functions, by using notions of subgradient and sub-differential, see [Nes18].

Problem formulation with a separable function The problem (2.1) can also be written using a separable function, defined in Definition 2.5 and referring to the concept of partially additively separable function in mathematics [Via08].

Definition 2.5 (Separable function)**.** We say that a function $F^{s_n} : \mathbb{R}^{nd} \to \mathbb{R}$ is *separable* if there are n functions $f_i : \mathbb{R}^d \to \mathbb{R}$ $(i = 1, \ldots, n)$ such that

$$F^{s_n}(x_1, \ldots, x_n) = \frac{1}{n}\sum_{i=1}^{n} f_i(x_i).$$

Using a separable function $F^{s_n} : \mathbb{R}^{nd} \to \mathbb{R}$, problem (2.1) can be written as

$$\begin{aligned} \min_{x_1 \ldots, x_n \in \mathbb{R}^d} \quad & F^{s_n}(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} f_i(x_i), \\ \text{such that} \quad & x_1 = \cdots = x_n. \end{aligned} \tag{2.4}$$

The local variables of all the agents can be stacked vertically in $\mathbf{x} \in \mathbb{R}^{nd}$:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

Using this notation, we now define the set of points $\mathbf{x} \in \mathbb{R}^{nd}$ satisfying the consensus constraint in Problem (2.4), called the consensus subspace.

Definition 2.6 (Consensus subspace)**.** The consensus subspace of $\mathbb{R}^{nd}$ is denoted $\mathcal{C}$ and is defined as

$$\mathcal{C} = \left\{ \mathbf{x} \in \mathbb{R}^{nd} \mid x_1 = \cdots = x_n \in \mathbb{R}^d \right\}.$$

The dimension of $\mathcal{C}$ is $|\mathcal{C}| = d$. The orthogonal complement of $\mathcal{C}$ is denoted $\mathcal{C}^\perp$ and has dimension $|\mathcal{C}^\perp| = (n-1)d$:

$$\mathcal{C}^\perp = \left\{ \mathbf{x} \in \mathbb{R}^{nd} \mid \mathbf{x}^T \mathbf{y} = 0, \text{ for all } \mathbf{y} \in \mathcal{C} \right\}.$$

Therefore, the feasible set of Problem (2.4) is simply given by the consensus subspace $\mathcal{C}$, and the problem becomes

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^{nd}} \quad & F^{s_n}(\mathbf{x}) \\ \text{s. t.} \quad & \mathbf{x} \in \mathcal{C}. \end{aligned}$$

This is why this decentralized optimization problem is called the *decentralized consensus optimization problem* or the *multi-agent consensus optimization problem*.

2.2 Relevance and applications

Many problems can be formulated as minimizing the sum of convex functions, where each function is accessible to only one agent. In such cases, it could be inefficient or even impossible for all agents to share their local functions with a master coordinator or with each other, often because the agents are too numerous, or the functions depend on large amounts of data. In addition, sometimes local functions cannot be shared for privacy reasons. Therefore, it is relevant to use decentralized optimization algorithms that require only local exchanges of solution estimates or other variables, instead of the full local functions. Popular fields of applications include multi-robot systems [CYRC12, HSYS21], sensor networks [RN04, PKP06, LT10], power systems [YTX13, YHL15, YLW+16, MDS+17] and machine learning [FCG10, MBG10, CHW+20, BPS+23]. These many useful applications have motivated intense research interest in decentralized consensus optimization over the last decade. We present a few of them below and refer to surveys [NL18, NOR18, HSYS21] for a more extensive overview.

2.2.1 Decentralized Estimation

In sensor networks or power system networks, it is often needed to estimate environmental or system parameters. While this can be achieved by sending all the data to a central node for processing, a distributed in-network processing is generally preferable as it can significantly decrease communication and energy consumption, as shown in [RN04]. For this purpose, agents can rely on decentralized consensus optimization algorithms, since parameter estimation consists in minimizing an estimation criterion f, which often has the following form

$$f(x) = \sum_{i=1}^{n} f_i(x),$$

where x is the parameter to estimate and $f_i(x)$ is a local function that only depends on the data measured by agent i. For example, the estimation criterion could be

- **Linear least squares**, in which case the local functions are given by

$$f_i(x) = \|A_i x - b_i\|^2,$$

where A_i, b_i are respectively a matrix and a vector held locally by agent i and built from its measurements. This has application, for example, in large power system networks, for estimating the electromechanical oscillation modes [NZC15], which are needed for maintaining system stability and performance.

- **Maximum likelihood**, in which case the local functions are given by

$$f_i(x) = -\sum_{j \in \mathcal{J}_i} \log\big(p(\eta_j|x)\big),$$

where $\{\eta_j\}_{j=1}^u$ denotes measurements of the agents, $\mathcal{J}_i$ the set of indices of the measurements of agent i, and $p(\eta_j|x)$ the conditional density probability function, which describes the probability of observing η_j when the parameter is given by x. Moreover, the measurements are assumed to be conditionally independent given the model parameter x, so that the total likelihood can be decomposed in a product as $p\big(\{\eta_j\}_{j=1}^u|x\big) = \prod_{j=1}^u p(\eta_j|x)$. The resulting problem is convex for any log-concave density function p, which is the case for many common probability distributions (e.g., normal, exponential, uniform, logistic, chi, beta). This approach is applicable, for instance, in wireless sensor networks deployed to monitor remote or hard-to-reach locations, where the estimate of x is used in scientific observations (e.g., bird migration patterns) or event detection (e.g., avalanches) [BISV08, IBS+10].

2.2.2 Wind farm control

There are various wind farm control problems [KBS15], which can be addressed with decentralized optimization, including maximizing the total power production of the wind farm. Turbines are often placed next to each other to reduce installation, operation, and maintenance costs. However, turbine operation creates wakes that can diminish wind speed and power output for downstream turbines. These aerodynamic interactions affect the overall performance of the wind farm. The greedy policy, where each turbine optimizes its own power output without considering its impact on downstream turbines, is often used in practice, but it generally leads to suboptimal power output for the whole wind farm. This can result in a power loss of over 30% in some worst-case scenarios [XCGN22]. Cooperation between turbines via decentralized consen-

sus optimization is therefore a promising alternative studied in the literature [GSS+19, XCGN22]. The distributed power maximization problem in a wind farm can be framed as

$$\max_{x} \quad \sum_{i=1}^{n} P_i(x),$$

where x is the decision vector with operating parameters of all the turbines and P_i is the power function of turbine i, taking into account the wake interactions with upstream turbines. The turbine operating parameters that are optimized can include the yaw angle (angle between wind direction and rotor axis), the blade pitch angle, and the generator torque of the turbine. These variables all influence the resulting power production and wake, and they generally have box constraints that need to be considered in the problem.

2.2.3 Machine learning

Supervised learning methods (e.g., classification or regression) can mostly be formulated as fitting a model to data. The task is generally to find the model parameters x that minimize the loss observed on a set of training data:

$$\min_{x} \sum_{j=1}^{u} l_j(x), \tag{2.5}$$

where the loss function $l_j(x)$ measures the error performed by the model on the training data j, using parameters x, and the training data set contains u instances in total. Different machine learning models have convex loss functions (e.g., linear, logistic, ridge regression, and support vector machines), but this is generally not the case of deep neural networks (e.g., multilayer perceptrons). Nevertheless, many empirical works show that algorithms for convex optimization perform well on deep neural networks (see e.g., [SC16]).

When the training data set is too large (for huge u), it may become impossible to store the data on a single server. It may also be desirable to partition the data across multiple nodes for other reasons, such as privacy concerns, geographic distribution, or to speed up training by exploiting parallel computing resources. In these cases, the training problem (2.5) can be solved using decentralized optimization with local objective functions

of the form:

$$f_i(x) = \sum_{j \in \mathcal{J}_i} l_j(x),$$

where $\mathcal{J}_i$ is the set of indices of training data stored at node i. Such distributed machine learning problems have been studied, for instance, in [FCG10, MBG10, LCZ22]. A popular approach for solving distributed machine learning problems is *Federated learning* [MMR+17] and relies on a central server that coordinates the agents, without accessing their local data sets. Federated learning algorithms can be seen as a specific distributed machine learning approach that only applies to a star communication graph, where one central agent is connected to all the others (see Fig. 1.1a). Recently, decentralized federated learning, which does not require any central node, has also been studied, see e.g., [HBJ18, LCZ22, BPS+23].

Decentralized versus Distributed

In this thesis, we make no distinction between the terms decentralized and distributed and use them interchangeably. Potential distinctions between the two terms in the literature have evolved over time [Ver20] and there is no clear scientific consensus. In [Ver20], the author proposes to distinguish decentralization, as the dispersion of organizational communications, from distribution, as the dispersion of organizational decision-making. While relevant in general, we will not apply such distinction in this thesis. We refer to decentralized (or distributed) algorithms as multi-agent algorithms that do not require a central node to make decisions or organize communications. In general, decentralized systems consider homogeneous agents, that have similar abilities and perform the same algorithm to solve the problem. When specified, we sometimes consider a certain heterogeneity in the set of agents, regarding their type of local function, their initial estimate of the solution, or the algorithm parameters. Moreover, the topology of the communication network is free and can, in particular, correspond to a full graph, linking all the nodes together, or to a star graph, in which only one node is connected to all the others.

2.3 Communications and average consensus

Interactions between the agents are essential to allow them to converge toward consensus on an optimal solution x^* of problem (2.1). They take place via a connected communication network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of agents and $\mathcal{E}$ their set of communications links. During a communication, the agents share one of their local variables with their neighbors. Then, the agents aggregate the variables received from their neighbors, usually via a weighted sum:

$$y_i = \sum_{i=1}^{n} w_{ij} x_j \qquad \text{for } i = 1, \dots, n, \tag{2.6}$$

where x_j can represent any vector in $\mathbb{R}^d$ held by agent j, vector $y_i \in \mathbb{R}^d$ is an auxiliary variable of agent i that represents the result of the interaction, and $W = [w_{ij}] \in \mathbb{R}^{n \times n}$ is a mixing matrix, matching the network topology and generally assumed doubly stochastic. Some decentralized algorithms rely on a constant mixing matrix, which is the same for each communication, while others allow time-varying mixing matrices, and so potentially also time-varying communication networks.

Update (2.6) corresponds to an average consensus step and can be repeated as

$$x_i^{k+1} = \sum_{i=1}^{n} w_{ij}^k x_j^k, \tag{2.7}$$

to converge to the initial agent average, under some conditions on the mixing matrices used, see Section 2.3.2. The average consensus is an important building block of decentralized optimization algorithms because of the nature of the decentralized problem (2.1). In this section, we first review a few possible types of communication settings, before providing details about the setting considered in this thesis. Then, we provide standard convergence results for the linear average consensus (2.7), and we finish by discussing a few other consensus-related work.

Directed vs Undirected When the communication network $\mathcal{G}$ is undirected, the communications are bidirectional and the mixing matrices associated with the graph are symmetric. Undirected communications are widely used and result in simpler algorithm analyses. However, many

practical applications involve directed graphs. For example, if the agents communicate through a wireless network, their communication radius depends on their broadcasting power. If agents use different power levels, the resulting communications will naturally be directed. When the communication network $\mathcal{G}$ is directed, this means that the communications are unidirectional, i.e., a node i can send messages to node j at iteration k without receiving any message from j at this iteration. In that case, the mixing matrices associated with the graph may be non-symmetric. The directed communication setting does not generally allow balancing weights to ensure convergence of the linear consensus step (2.7) towards the average, and requires more advanced techniques to achieve average consensus between agents, such as the push-sum protocol [KDG03].

Synchronous vs Asynchronous In synchronous algorithms, all agents perform their updates and communications in a synchronized way. This ensures that they all perform the same number of updates. Synchronous algorithms are simpler to analyze but require a common clock in the system. Moreover, if there are differences in the computational power of the agents, the faster agent will wait for the slowest ones, which are often called *stragglers*. Asynchronous algorithms resolve this problem by allowing the nodes to perform their updates and communications independently of each other. Practically, for the analysis, one may resort to a random pairwise gossip communication [BGPS06, DKM+10] where a random link is activated for communication or to a random broadcast communication [AYSS09] where a random node is activated to broadcast its information to its neighbors.

2.3.1 Setting considered: synchronous and undirected communications

In this thesis, we only consider interactions through synchronous and undirected communications, that take the form of a linear average consensus on some quantity, i.e.,

$$y_i = \sum_{j=1}^{n} w_{ij} x_j, \qquad \text{for } i = 1, \dots, n.$$

The mixing matrix $W = [w_{ij}] \in \mathbb{R}^{n \times n}$ is assumed to be (i) symmetric, since the communications are undirected, (ii) *generalized doubly stochastic* [HL05], i.e., with rows and columns summing to one, to allow average consensus,

and (iii) matching the network topology. We call such matrices *averaging matrices*.

Definition 2.7. We say that a matrix $W \in \mathbb{R}^{n\times n}$ is an *averaging matrix* if

(i) $W^T = W$, (Symmetric)

(ii) $W\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T W = \mathbf{1}^T$, (Generalized doubly stochastic)

(iii) $W \in \mathcal{T}$, (Network topology)
where $\mathcal{T} = \{W : w_{ij} = 0 \text{ if } (i,j) \notin \mathcal{E} \text{ and } i \neq j\}$.

An averaging matrix $W \in \mathbb{R}^{n\times n}$ can be considered as an adjacency matrix of a weighted version of graph $\mathcal{G}$, with possible self-loops.

Remark 2.8 (On the assumptions for the mixing matrices).

- *Symmetry:* This thesis focuses on symmetric averaging matrices, which assume undirected communication links, but our results could be extended to non-symmetric ones, by working with singular values instead of eigenvalues.

- *Possible negative elements:* We do not restrict to non-negative matrices, even if the literature of decentralized optimization often assumes non-negativity through doubly stochasticity. We motivate this choice by two arguments. Firstly, non-negativity is not needed for convergence of pure linear consensus steps [XB04], see Theorem 2.9 below. Secondly, among results of decentralized optimization using doubly stochasticity, we found no results exploiting the non-negativity of the matrix, see e.g., [NOR18, NOS16, SLWY14, LSY17]. Such results are thus about generalized doubly stochastic matrices.

2.3.2 Consensus convergence results

The convergence of a linear average consensus protocol (2.7) towards the agent average can be characterized by the eigenvalues of the mixing matrices used. This statement is detailed in Theorem 2.9 and Corollary 2.10 below, for a time-invariant mixing matrix $W \in \mathbb{R}^{n\times n}$ and for scalar local variables $x_i \in \mathbb{R}$. Equivalent results also hold for vectors of dimensions d, i.e., $x_i \in \mathbb{R}^d$, but require a more complex notation, detailed in Section 2.7. The eigenvalues of matrix W are denoted by $\lambda_1(W), \ldots, \lambda_n(W)$, in descending order of their real parts. Convergence of the average consensus

with time-varying matrices, and so potentially also time-varying communication networks, is analyzed e.g., in [BHOT05, OT09] and is briefly discussed in Section 2.3.3.

Theorem 2.9 (Average consensus convergence [XB04])**.** *Let $W \in \mathbb{R}^{n\times n}$ and $\mathbf{x}^0 \in \mathbb{R}^n$ gather the initial scalar values $x_i^0 \in \mathbb{R}$ of each agent $i = 1,\dots,n$. The distributed linear consensus $\mathbf{x}^{k+1} = W\mathbf{x}^k$ converges to the agent average $\overline{\mathbf{x}} = \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}^0$ if and only if the matrix W satisfies*

(i) $W\mathbf{1} = \mathbf{1}$ *and* $\mathbf{1}^T W = \mathbf{1}^T$,

(ii) $\lambda = \rho\big(W - \frac{\mathbf{1}\mathbf{1}^T}{n}\big) < 1$,

where $\rho(\cdot) = \max\{|\lambda_1(\cdot)|,\dots,|\lambda_n(\cdot)|\}$ denotes the spectral radius of a matrix. Moreover, the asymptotic convergence rate of $\mathbf{x}^k$ toward $\overline{\mathbf{x}}$ is given by λ:

$$\sup_{\mathbf{x}^0\neq\overline{\mathbf{x}}} \lim_{k\to\infty} \left(\frac{\|\mathbf{x}^k - \overline{\mathbf{x}}\|_2}{\|\mathbf{x}^0 - \overline{\mathbf{x}}\|_2}\right)^{1/k} = \rho\Big(W - \frac{\mathbf{1}\mathbf{1}^T}{n}\Big) = \lambda,$$

and the per-step convergence rate is given by:

$$\sup_{\mathbf{x}^0\neq\overline{\mathbf{x}}} \frac{\|\mathbf{x}^{k+1} - \overline{\mathbf{x}}\|_2}{\|\mathbf{x}^k - \overline{\mathbf{x}}\|_2} = \Big\|W - \frac{\mathbf{1}\mathbf{1}^T}{n}\Big\|_2,$$

where $\|\cdot\|_2$ denotes the spectral norm of the matrix, i.e., its maximum singular value.

The proof of Theorem 2.9 can be found in [XB04, Theorem 1].
When the matrix W is a symmetric averaging matrix, condition (i) is satisfied by Definition 2.7, and the two convergence rates coincide since the singular values of a symmetric matrix correspond to its absolute eigenvalues:

$$\lambda = \rho\Big(W - \frac{\mathbf{1}\mathbf{1}^T}{n}\Big) = \Big\|W - \frac{\mathbf{1}\mathbf{1}^T}{n}\Big\|_2.$$

Therefore, we can simplify the result of Theorem 2.9 as follows

Corollary 2.10. *Let $W \in \mathbb{R}^{n\times n}$ be a valid symmetric averaging matrix for a given graph $\mathcal{G}$ (Definition 2.7) and let $\mathbf{x}^0 \in \mathbb{R}^n$ gather the initial scalar values $x_i^0 \in \mathbb{R}$ of each agent $i = 1,\dots,n$. The distributed linear consensus $\mathbf{x}^{k+1} = W\mathbf{x}^k$ converges to the agent average $\overline{\mathbf{x}} = \frac{\mathbf{1}\mathbf{1}^T}{n}\mathbf{x}^0$ if and only if*

$$\lambda = \|W - \frac{\mathbf{1}\mathbf{1}^T}{n}\|_2 < 1.$$

Moreover, the convergence rate of $\mathbf{x}^k$ toward $\overline{\mathbf{x}}$ is given by λ.

By definition, an averaging matrix W always has 1 as eigenvalue, associated with eigenvector $\mathbf{1}$. When $\lambda = \|W - \frac{\mathbf{1}\mathbf{1}^T}{n}\|_2 < 1$, we know that 1 corresponds to the largest eigenvalue of W, i.e., $\lambda_1(W) = 1$, and λ gives the Second Largest Eigenvalue Modulus (SLEM) of W:

$$\lambda(W) = \left\|W - \frac{\mathbf{1}\mathbf{1}^T}{n}\right\|_2 = \max\left\{|\lambda_2(W)|, |\lambda_n(W)|\right\}. \tag{2.8}$$

In graph theory, the SLEM of a non-negative averaging matrix W is related to the algebraic connectivity $\gamma(W)$ of the corresponding weighted graph, defined as the spectral gap of W

$$\gamma(W) = 1 - \lambda(W).$$

The algebraic connectivity γ is usually defined as the second-smallest eigenvalue of the (weighted) Laplacian matrix $L = I - W$, see e.g., [Fie89, DA07]. The value of γ characterizes, in some sense, the connectivity of the resulting weighted graph, and in particular, it can be used to bound the edge and vertex connectivity [Fie89, DA07].

Example 2.11. When $\gamma = 1$, we have $\lambda = 0$. This corresponds to a perfectly connected and balanced weighted graph, associated with the averaging matrix $W = \frac{\mathbf{1}\mathbf{1}^T}{n}$, which has $\lambda_1 = 1$ and $\lambda_i = 0$ for $i = 2, \ldots, n$.

Here is a well-known proposition showing a link that exists between the SLEM of matrix W (2.8) and the connectivity of its underlying graph. Recall that an undirected graph is said to be connected if there is a path between every pair of vertices in the graph.

Proposition 2.12. *Let W be an averaging matrix (Definition 2.7) and $\mathcal{G}_W$, its underlying weighted graph. The following implication holds*

$$\lambda(W) < 1 \quad \Rightarrow \quad \mathcal{G}_W \text{ is connected}$$

Moreover, if W is non-negative, then the reverse implication also holds, meaning that the two conditions are equivalent:

$$\lambda(W) < 1 \quad \Leftrightarrow \quad \mathcal{G}_W \text{ is connected}$$

Proof. We prove the first statement by showing that an averaging matrix W associated with a disconnected $\mathcal{G}_W$ always has its second-largest eigenvalue modulus equal to one: $\lambda(W) = 1$. An averaging matrix W associated

with a disconnected $\mathcal{G}_W$ has at least 2 distinct connected components and can thus be written with at least 2 blocks on the diagonal:

$$W = \begin{bmatrix} W_1 & 0 \\ 0 & W_2 \end{bmatrix}.$$

Since $W\mathbf{1} = \mathbf{1}$, we also have $W_u\mathbf{1}_u = \mathbf{1}_u$ for $u = 1, 2$, and vector $\mathbf{1}_u$ of appropriate size. Therefore, the matrix W has (at least) two independent eigenvectors associated with eigenvalues 1:

$$v_1 = \begin{bmatrix} 1 & \dots & 1 & 0 & \dots & 0 \end{bmatrix}^T \quad \text{and} \quad v_0 = \begin{bmatrix} 0 & \dots & 0 & 1 & \dots & 1 \end{bmatrix}^T.$$

This means that the two largest eigenvalues of W are equal to 1, and so $\lambda(W) = 1$.

The second statement is well-known in graph theory and follows from the fact that the multiplicity of the 0 eigenvalues of the Laplacian matrix $L = I - W$ corresponds to the number of connected components of the underlying graph, see e.g., [GR01] for a proof. □

The first statement of Proposition 2.12 means that for averaging matrices (that can have negative entries), having $\lambda < 1$ guarantees that the underlying graph is connected, but the converse is not true since there are connected weighted graphs for which $\lambda(W) = 1$, e.g., the graph associated with the following matrix

$$W = \begin{bmatrix} -1/3 & 2/3 & 2/3 \\ 2/3 & -1/3 & 2/3 \\ 2/3 & 2/3 & -1/3 \end{bmatrix}.$$

Finally, we can also combine Proposition 2.12 and Corollary 2.10 to claim that any non-negative averaging matrix W (Definition 2.7) associated with a connected graph $\mathcal{G}$ guarantees the convergence of the pure linear consensus to the average, at a rate given by λ (2.8).

2.3.3 Related work

We close this section by providing some details and references on other related work relevant to the field of average consensus and distributed algorithms, that has not been discussed previously.

Time-varying communication networks In the literature of consensus, time-varying graphs refer to a graph sequence $\mathcal{G}_0, \mathcal{G}_1, \ldots$, in which each graph instance $\mathcal{G}_k$ has a time-invariant set of agents $\mathcal{V}$ and a time-varying set of edges $\mathcal{E}_k$. Then, iteration k of a consensus protocol relies on communication network $\mathcal{G}_k$. In the undirected case, agents i and j can exchange information at iteration k if and only if they are linked by an edge in $\mathcal{G}_k$, i.e., $(i, j) \in \mathcal{E}_k$. Convergence results with time-varying graphs generally rely on the assumption that the graph sequence is B-connected, for some constant $B \geq 1$, see e.g., [JLM03, OT09, NOOT09, CI14, LJ11, SRNV12, NOS16].

Definition 2.13 (B-connected graph sequence)**.** A time-varying undirected graph sequence $\mathcal{G}^0, \mathcal{G}^1, \ldots$ is B-connected if each subsequence of B successive graphs is jointly connected, i.e., their union is connected:

$$\cup_{l=k}^{l=k+B-1} \mathcal{G}_l \quad \text{is connected} \quad \text{for all } k = 0, 1, \ldots.$$

This definition extends to directed graphs, by replacing the term "connected" by "strongly connected". Recall that a directed graph is strongly connected if there is a path between each pair of vertices of the graph, in both directions. The use of time-varying averaging graphs is useful, for example, in multi-robot systems applications where the agents are moving in the space and therefore have time-varying neighbors to communicate with [JLM03].

Open multi-agent systems Some work has also been done in the case where the set of agents can also vary over time [VMdGF+24], which occurs in applications with frequent departures and arrivals of agents such as social network, multi-robot systems, and epidemics [VFP24]. These systems are known as *open multi-agent systems* [VMdGF+24]. When the set of agents is constantly evolving, the consensus between the agents in the system may never be reached [MdGH22].

Another convergence characterization The convergence rate of the average consensus (2.7) can also be characterized based on the number of agents and a lower bound on the nonzero weights used in the algorithm, see e.g., [NOOT09]. In [NOOT09], the authors prove that the worst-case convergence time of the linear average consensus (2.7) with time-varying averaging matrices scales quadratically with the number of agents n and that this dependence is tight. While the graph topology can generally not

be chosen, we can choose the weights used during the communications to improve the rate dependence on n [XB04, NOOT09, NOR18]. Moreover, when the agents all know a common upper bound U on the number of agents in the network, [Ols17] proposes another consensus algorithm for which the worst-case convergence time scales linearly with n. This accelerated consensus algorithm relies on local degree information to choose the communication weights in the linear consensus (2.7) and add a momentum step which exploits the value of U.

Other related work There are many interesting and relevant works in the field of average consensus and distributed algorithms, e.g., related to privacy issues [MM16, GKC17, RAW17], faulty or malicious agents [PBB11], packet losses [FH13, GH18, XMX19], communication noise [XBK07, KM08, LZ10, RW10], communication time-delays [OSM04, BHOT05, NO10, LQZS12, CYY+15] and quantized or compressed communications [RN05, ACR08, NOOT09, FCFZ09, KSJ19, TU22].

2.4 Existing algorithms for decentralized optimization

This section reviews the algorithms that exist to solve the decentralized problem (2.1). We provide a few details on the main or historical methods, which have inspired many others. We refer to [YYW+19] and [GRB+22] for more detailed surveys.

2.4.1 DGD

One of the first methods proposed to solve (2.1) is the distributed (sub)gradient descent (DGD) [NO09] where agents successively perform an average consensus step (2.9) and a local gradient step (2.10), as described in Algorithm 1.

Algorithm 1 Distributed Gradient Descent (DGD)

Input: $x_i^0 \in \mathbb{R}^d$, step-sizes $\alpha^k > 0$ and averaging matrices $W^k \in \mathbb{R}^{n \times n}$,
For $k = 0, 1, \ldots$

$$y_i^k = \sum_{j=1}^{n} w_{ij}^k x_j^k, \qquad \text{for } i = 1, \ldots, n, \tag{2.9}$$

$$x_i^{k+1} = y_i^k - \alpha^k \nabla f_i(x_i^k), \qquad \text{for } i = 1, \ldots, n, \tag{2.10}$$

Convergence in a neighborhood The DGD algorithm is simple and intuitive, but it is only converging to a neighborhood of the optimal solution when the step-size is constant [YLY13]. In particular, the optimal solution x^* for (2.1) is not a stationary point for the update (2.9)-(2.10) since the local gradients can be non-zero at the optimum, i.e., $\nabla f_i(x^*) \neq 0$.

Convergence with diminishing step-sizes Convergence to the optimum can be guaranteed for decreasing step-size sequences and (possibly) non-smooth convex local functions [SRNV12, NOR18]. The proof holds for time-varying mixing matrices which are doubly stochastic, associated with a B-connected graph sequence. In theory, convergence holds for directed graphs, but in practice, directed communications do not generally allow balancing weights to have doubly stochastic mixing matrices. To obtain a converging algorithm over time-varying directed graphs, without the doubly stochasticity assumption, the authors of [NO14] combine the DGD algorithm with the push-sum consensus protocol [KDG03], and prove the convergence of their resulting *subgradient-push method* with appropriate diminishing step-sizes.

Convergence of the running average Alternatively, DGD with a constant step-size requires considering the running average iterate, i.e., the average over all agents and all iterations, to obtain an error going to zero [NO09, NOR18] in similar settings.

Variants and extensions The asynchronous variant of DGD has been studied in [NO09, Ned10] and its stochastic variant in [SRNV10].

Then, other algorithms have been developed to tackle the DGD convergence problem with constant step-sizes. Their strategy, to converge to the

optimal solution x^* with a constant step-size, is generally based on the use of historical information. This is the case of EXTRA [SLWY14] and DIGing [NOS16], which we present below.

2.4.2 EXTRA

The method has been introduced in [SLWY14] as the *exact first-order algorithm* and abbreviated as EXTRA. The method, described in Algorithm 2, is built as the difference between two consecutive DGD updates, using averaging matrices W and $\tilde{W}$, to cancel their remaining error. In particular, in [SLWY14], the authors showed that EXTRA can be seen as DGD with a cumulative correction term to compensate for the error that appears in DGD with a constant step-size. The resulting update (2.11) requires the two past iterates and local gradients to compute x_i^{k+1}.

Algorithm 2 EXTRA

Input: $x_i^0 \in \mathbb{R}^d$ $(\forall i)$, step-size $\alpha > 0$, averaging matrices W, $\tilde{W} \in \mathbb{R}^{n\times n}$

$$x_i^1 = \sum_{j=1}^{n} w_{ij} x_j^0 - \alpha \nabla f_i(x_i^0), \qquad \text{for } i = 1, \dots, n,$$

For $k = 1, 2, \dots$ and $i = 1, \dots, n$,

$$x_i^{k+1} = x_i^k + \sum_{j=1}^{n} \left(w_{ij} x_j^k - \tilde{w}_{ij} x_j^{k-1} \right) - \alpha \left(\nabla f_i(x_i^k) - \nabla f_i(x_i^{k-1}) \right). \quad (2.11)$$

Authors in [SLWY14] recommend choosing $\tilde{W} = \frac{W+I}{2}$. This allows computing only one new consensus step at each iteration.

Convergence For an undirected connected network and under some conditions on the constant averaging matrices and step-size, [SLWY14] proved that EXTRA has a convergence in $\mathcal{O}(\frac{1}{k})$ for smooth convex local functions and a linear convergence for L-smooth and (restricted) μ-strongly convex local functions.

Accelerated EXTRA The EXTRA algorithm can also be interpreted as a primal-dual algorithm using the augmented Lagrangian function [MR16, HHZ17, LL20]. The authors in [LL20] leverage this form of the algorithm to sharpen its performance analysis and provide an accelerated version of EXTRA using the Catalyst framework [LMH18].

Composite convex distributed problems The EXTRA algorithm has also been extended to composite convex problems where agents hold local functions with smooth and non-smooth convex composites, resulting in the proximal gradient exact first-order algorithm (PG-EXTRA) [SLWY15]. Such composite problems are particularly relevant for machine learning applications where the regularization terms are often non-smooth.

Other decentralized algorithms have been designed for solving such composite convex distributed problems, see e.g., NIDS [LSY17], P2D2 [AYS19], SONATA [SSD22], ProxMudag [YLZZ23], or the unified framework and analysis [XTSS21]. Any method working for composite convex distributed problems also work for the standard convex distributed problem, by setting the non-smooth composite to zero.

2.4.3 DIGing

This algorithm has been developed independently in [NOS16] and [QL17]. It combines the distributed inexact gradient method (DGD) with the gradient tracking technique and is abbreviated as DIGing. We describe it in Algorithm 3. In the algorithm, each agent i, in addition to its estimate x_i of the optimal solution x^*, also maintains a second state variable s_i which estimates the average gradient. Each agent updates its solution estimate x_i with an update similar to DGD, but using the gradient estimate s_i instead of their own local gradient (2.12). The second update (2.13) tracks the average gradient using dynamic average consensus [ZM10]. The resulting algorithm requires the agents to communicate two variables (x_i and s_i) at each step.

Algorithm 3 DIGing

Input: $x_i^0 \in \mathbb{R}^d$, $s_i^0 = \nabla f_i(x_i^0)$, $\alpha > 0$ and averaging matrices $W \in \mathbb{R}^{n\times n}$
For $k = 0, 1, \ldots$

$$x_i^{k+1} = \sum_{j=1}^{n} w_{ij}^k x_j^k - \alpha s_i^k, \qquad \text{for } i = 1, \ldots, n, \tag{2.12}$$

$$s_i^{k+1} = \sum_{j=1}^{n} w_{ij}^k s_j^k + \left(\nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k)\right). \qquad \text{for } i = 1, \ldots, n, \tag{2.13}$$

Convergence In [QL17], the authors established the linear convergence of DIGing over undirected connected networks, with a constant averaging

matrix and a constant step-size, under the assumption that local objective functions are smooth and strongly convex. Then, in [NOS16], the authors extend the convergence result to time-varying averaging matrices, and so time-varying communication networks. Moreover, they only assume that the sequence of graphs is B-connected (see Definition 2.13).

Relation between DIGing and EXTRA For constant undirected networks, [NOS16] showed that the DIGing algorithm is a particular case of the EXTRA algorithm, with a specific choice for the two averaging matrices in EXTRA. Moreover, they also gave a primal-dual interpretation of the DIGing algorithm based on the augmented Lagrangian function. In [Jak18], the authors proposed a unified primal-dual analysis of EXTRA and DIGing and revealed that an important difference between these two methods is on how the primal error affects the dual error. They used this insight to generalize these methods by deriving a new method that reduces the negative effect of primal error on the dual error.

Push-DIGing In [NOS16], the authors also adapt the algorithm to directed communication graphs, by combining it with the push-sum consensus protocol [KDG03]. The resulting algorithm is called Push-DIGing.

ATC-DIGing The algorithm proposed in [NOSU17, XZSX15] corresponds to the *adapt-then-combine* (ATC) version of DIGing:

$$x_i^{k+1} = \sum_{j=1}^{n} w_{ij}^k \left(x_j^k - \alpha s_j^k \right), \qquad \text{for } i = 1, \dots, n, \quad (2.14)$$

$$s_i^{k+1} = \sum_{j=1}^{n} w_{ij}^k \left(s_j^k + \nabla f_j(x_j^{k+1}) - \nabla f_j(x_j^k) \right). \qquad \text{for } i = 1, \dots, n, \quad (2.15)$$

ATC-DIGing differs from DIGing only in the order of operations: it first adapts the variables with local information, then combines them with information from other agents. Distributed algorithms generally require agents to coordinate in advance, so that they all use the same step-size α for their updates, but the ATC-DIGing allows working with uncoordinated constant step-sizes $\alpha_1, \dots, \alpha_n$, while preserving the linear convergence of the algorithm. In [NOSU17], they also observed that the ATC structure can accelerate the convergence compared to the *combien-then-adapt* (CTA) structure used in DIGing or DGD. This observation will be validated by our analysis in Chapter 8.

Gradient tracking There are many other recent decentralized optimization algorithms that rely on the gradient tracking technique, such as the Accelerated Distributed Nesterov Gradient Descent (Acc-DNGD) [QL19], the Accelerated Gradient Tracking (Acc-GT) [LL21] and the Optimal Gradient Tracking method (OGT) [SSPY23].

2.4.4 Dual-based methods

Primal methods directly focus on solving the original primal problem (2.4) without explicitly considering any associated dual problem. The consensus constraint in the primal problem (2.4) is linear and allows for a dual reformulation of the problem. Dual methods introduce the Lagrangian function in the objective and work in the dual space to solve the dual problem. The DGD, EXTRA, and DIGing algorithms are explicit primal methods, that combine gradient sampling and consensus steps. However, EXTRA and DIGing can also be written in a primal-dual form, meaning they simultaneously optimize both the primal and dual problems, and their reformulated updates involve primal and dual variables. By the term *dual-based* methods, we mean both primal-dual methods and dual methods, where the use of the dual problem and dual variables is an integral part of the algorithm. Distributed dual-based algorithms include the dual ascent [TTM11] and its accelerated version [ULGN20], the distributed dual averaging (DDA) [DAW12], the multi-step dual accelerated (MSDA) [SBB+17], the multi-step primal-dual (MSPD) [SBB+18], the decentralized communication sliding (DCS) [LLZ20], the accelerated decentralized optimization method (ADOM) [KSR+21, KGGR21] and the distributed version of the alternating direction method of multipliers (D-ADMM) [SLY+13, BPC+11, IBCH15, LLST16, MO17, AWLM17]. Many dual-based methods have a larger computation cost per iteration since they involve a gradient evaluation of the Fenchel conjugate of the local objective function or its proximal mapping.

2.4.5 Accelerated and optimal methods

In general, optimization theory evaluates the efficiency of an algorithm by counting the number of iterations needed in the worst-case to obtain a guaranteed ϵ-accuracy on the optimal solution. While this is also done in decentralized optimization, the literature often distinguishes the gradient computation complexity from the communication complexity. The

gradient computation complexity counts the number of local gradients computed by each agent, to achieve the desired accuracy. The communication complexity counts the number of communication rounds needed to achieve the desired accuracy. These complexities of course depend on the required accuracy ϵ, but also on the conditioning of the problem, i.e., the conditioning of the global or local functions and some characterization of the network connectivity, such as its SLEM λ. Over the last years, effort has been deployed to identify the lower complexity bounds in different settings [SBB+19, KGGR21] (see Table 2.1), as well as accelerated algorithms matching or approaching these lower bounds.

Settings	Gradient computation complexity [Nes18]	Communication complexity [SBB+19]
L-smooth μ-strongly convex	$\Omega\left(\sqrt{\frac{L}{\mu}}\log\left(\frac{1}{\epsilon}\right)\right)$	$\Omega\left(\sqrt{\frac{L}{\mu(1-\lambda)}}\log\left(\frac{1}{\epsilon}\right)\right)$
L-smooth convex	$\Omega\left(\sqrt{\frac{L}{\epsilon}}\right)$	$\Omega\left(\sqrt{\frac{L}{\epsilon(1-\lambda)}}\log\left(\frac{1}{\epsilon}\right)\right)$
Nonsmooth convex	$\Omega\left(\frac{1}{\epsilon^2}\right)$	$\Omega\left(\frac{1}{\epsilon(1-\lambda)}\right)$

Table 2.1 Lower bounds complexities for decentralized optimization algorithms over time-invariant graphs, to reach ϵ accuracy in terms of functional error [Nes18, SBB+19], i.e., $f(x_i^k) - f(x^*) \leq \epsilon$ for all $i = 1, \ldots, n$.

Smooth and strongly convex functions Many accelerated decentralized optimization methods have been developed for L-smooth and μ-strongly convex functions. To improve the rate dependence on function conditioning, accelerated methods generally relies on the Nesterov's acceleration technique, either applied in the primal space, see e.g., Acc-DNGD [QL19], APAPC [KSR20], Acc-GT [LL21], Mudag [YLZZ23], APD-SC [SSPY24], or in the dual space, e.g., ADA [ULGN20], MSDA [SBB+17], IDEAL [ABC+20], Acc-EXTRA [LL20], ADOM [KSR+21]. To improve the rate dependence on network conditioning, most methods relies on an inner loop with multiple communications rounds: either by applying the same averaging matrix multiple times, see e.g., [VSL19] or DAGD-C [RLG+21], by applying an accelerated average consensus (FastMix), see e.g., Mudag [YLZZ23] or APM-C [LFYL20], or relying on Chebyshev polynomials to cleverly adapt the communication weights in the inner loop,

see e.g., MSDA [SBB+17] or OPAPC [KSR20]. When the graph is time-varying, [LL21] and [KGGR21] developed an adapted multi-consensus scheme (MC). In [SSPY23], the authors proposed an Optimal Gradient Tracking (OGT) method which relies on a loopless Chebyshev acceleration technique and does not require multiple communication steps at each iteration. The method is called optimal because it matches the lower complexity bounds, for both the gradient computations and the communication complexities, that have been established in [SBB+17] for smooth and strongly convex functions and time-invariant graphs (see Table 2.1). MSDA [SBB+17], OPAPC [KSR20], Acc-GT+MC [LL21] and ADOM+MC [KGGR21] also matches these lower bounds. However, MSDA and ADOM+MC may be difficult to implement in practice because they require the agents to evaluate the gradients of local Fenchel conjugates at every iteration. In [KGGR21] the authors also propose a primal version of their algorithm ADOM+MC, but which involves 10 different updates and 12 parameters. Algorithm Acc-GT+MC is based on the gradient tracking technique, which allows convergence with time-varying graphs, and it matches the lower complexity bounds provided in [KGGR21] for time-varying connected graphs.

Smooth and convex Accelerated algorithms that can be applied to smooth and convex, but non-strongly convex, functions include [CO12], D-NG and D-NC [JXM14], Acc-DNGD [QL19], OPTRA [XTSS20], APM-C [LFYL20], Acc-EXTRA [LL20], Accelerated Dual Ascent (ADA) [ULGN20] and Acc-GT [LL21]. When functions are smooth and convex, the lower bound for gradient computation complexity can be deduced from centralized optimization theory [Nes18] and the lower bound for communication complexity is established in [SBB+19] (see Table 2.1). The Acc-DNGD [QL19] algorithm was the first accelerated method based on the gradient tracking idea. In [QL19], the authors have only proved a $\mathcal{O}(\frac{1}{\epsilon^{5/7}})$ complexity for Acc-DNGD, but they conjectured that its parameters can be chosen to obtain a $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$ complexity, which would match the lower bound for the gradient computations. This conjecture will be investigated in Chapter 8. Then, [LL21] proposed Acc-GT, another accelerated method based on gradient tracking. Acc-GT is equivalent to Acc-DNGD for convex functions ($\mu = 0$) and time-invariant graphs, and it matches the lower complexity bounds when combined with a multi-consensus inner loop. Currently, Acc-GT+MC [LL21] and OPTRA [XTSS20] are the only algorithms

matching both lower bounds and they both use an inner loop to accelerate the consensus. However, OPTRA only applies to time-invariant graphs and its convergence rate has only been proven in the Bregman distance metric.

Non-smooth and convex In general, results for non-smooth and convex functions assume that the local functions are Lipschitz continuous, i.e., with bounded subgradients. In these settings, the function properties cannot be exploited to accelerate the optimization component of the algorithms, but some effort can still be made to make the algorithms communication-efficient. In particular, this has been done in primal-dual algorithms, e.g., DCS [LLZ20], MSPD [SBB+18], and others such as APM [LFYL20]. These methods all include a proximal step or an inner loop of subgradient steps. Another inner loop of communication rounds is employed in MSPD to further reduce their complexity via the Chebyshev acceleration. The lower complexity bounds for these settings have been established in [SBB+18] (see Table 2.1) and are reached by MSPD and APM.

2.4.6 Other related work

Methods with directed communications As explained for DGD, many methods work for directed graphs, provided that the mixing matrices used are doubly stochastic. However, in practice, directed communications do not allow easily balancing of mixing weights to get doubly stochastic matrices. Therefore, *directed communications* generally refer to the case where communication graphs are directed and unbalanced, i.e., mixing matrices are not doubly stochastic. In that case, classical decentralized algorithms are generally not working, but can be adapted by changing the consensus protocol used, see subgradient-push [NO14] and Push-DIGing [NOS16], DEXTRA [XK17] and their accelerated versions [XXK18], [XXK18] and [SSPY24]. All these algorithms rely on the push-sum consensus protocol [KDG03]. Besides these algorithms, we also mention the Push-Pull gradient methods [PSXN20] and the AB/Push-Pull framework [XPNK20] that allow unifying different computation and communication architectures, as well as different algorithms, and that work both for undirected and directed graphs, synchronous and asynchronous settings.

Methods handling constraints In this thesis, we focus on the *unconstrained* decentralized optimization problem (2.1). However, many practical applications, such as motion planning or distributed resource allocations, involve constraints. We can define a (convex) constrained decentralized optimization problem as follows

$$\min \quad f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \qquad \text{such that} \quad x \in \mathcal{X} = \bigcap_{i=1}^{n} \mathcal{X}_i, \tag{2.16}$$

where f_i is the real convex local function of agent i and $\mathcal{X}_i$ its closed convex set, representing the local constraint set of agent i. When all agents hold the same closed convex constraint set, i.e., $\mathcal{X}_i = \mathcal{X}$ for all i, the existing algorithms can be modified by adding a projection on $\mathcal{X}$ in the local updates for the solution estimates. For example, DGD (Algorithm 1) can be modified with the following update:

$$x_i^{k+1} = \Pi_{\mathcal{X}}\left[\sum_{i=1}^{n} w_{ij}^k x_j - \alpha^k \nabla f_i(x_i^k)\right], \tag{2.17}$$

where $\Pi_{\mathcal{X}}[\cdot]$ is the Euclidean projection on the set $\mathcal{X}$. Since the projection operator is non-expansive, meaning $\|\Pi_{\mathcal{X}}[x] - \Pi_{\mathcal{X}}[y]\|_2 \leq \|x - y\|_2$ for all $x, y \in \mathbb{R}^d$, the convergence properties obtained for the unconstrained case are still applicable [NOP10]. When the constraint sets $\mathcal{X}_i$ are different, update (2.17) can be modified by replacing $\Pi_{\mathcal{X}}[\cdot]$ with $\Pi_{\mathcal{X}_i}[\cdot]$, but convergence analysis is less straightforward. This distributed projected (sub)gradient method has been introduced and analyzed in [NOP10]. Other works, developing new methods for solving the constrained problem (2.16), or performing convergence analysis of existing methods, include [ZM11, SN11, LN15, YHX15, LRS16, LYH17].

Stochastic methods In stochastic distributed optimization, the local gradients are computed with stochastic errors. Stochastic methods are very popular for data and machine learning applications. The main algorithms presented above have their stochastic variants: The stochastic variant of DGD has been studied in [SRNV10, KLB+20], the stochastic variant of DIGing in [PN21] and the stochastic variant of EXTRA in [MR16].

Methods for non-convex problems When the local objective functions are non-convex, it may be difficult to converge toward a global mini-

mizer of the decentralized problem, but different algorithms guarantee convergence toward a critical point or a local minimum for smooth non-convex local functions, such as the in-network successive convex approximation algorithm (NEXT) [DLS16], the proximal primal-dual algorithm (Prox-PDA) [HHZ17], which corresponds to a primal-dual formulation of EXTRA, the K-GT algorithm [LLKS24] and the SONATA algorithm [SS19]. Convergence of the *classical* decentralized algorithms in non-convex settings has also been studied in the literature, see e.g., [DSK20, CN22] for DIGing and [HHZ17] for EXTRA. We refer to [CHW+20] for a review of distributed optimization with non-convex functions, with a focus on data and machine learning applications.

2.5 The class of decentralized optimization methods considered

Besides DGD, EXTRA, and DIGing, there are many other methods combining gradient sampling and consensus steps. We call $\mathcal{A}_D$ the class of all these decentralized optimization algorithms.

Definition 2.14 (Class of decentralized optimization methods $\mathcal{A}_D$)**.** We define $\mathcal{A}_D$ as the class of all the decentralized optimization algorithms built based on the three following types of operations, which may involve an arbitrary number of local variables:

(i) *Gradient:* Each agent samples the (sub)gradient of its local function at any of its local variables x_i,

$$g_i = \nabla f_i(x_i) \qquad \text{for } i = 1, \dots, n.$$

(ii) *Consensus:* All the agents perform a consensus step on any of their local variables x_i,

$$y_i = \sum_{j=1}^{n} w_{ij} x_j, \quad \text{for } i = 1, \dots, n. \tag{2.18}$$

with weights w_{ij} given by an averaging matrix W (see Definition 2.7). Different consensus steps can use different averaging matrices, or possibly the same. .

(iii) *Linear Combinations:* Each agent declares that a linear combination of its local variables holds, with coefficients known in advance, e.g.,

$$x_i = y_i - \alpha g_i,$$

where x_i, y_i and g_i are local variables of agent i and α is a given linear coefficient (also called step-size). Depending on the settings, the linear coefficients (e.g. α) should be identical for all agents (coordinated) or not (uncoordinated).

Remark. In Definition 2.14, the vector $x_i \in \mathbb{R}^d$ refers to any local variable held by agent i, such as one of its local iterates, its local gradients or any linear combination of its local variables. Moreover, it can refer to a different local variable from one operation to the other. This makes $\mathcal{A}_D$ a very general class of decentralized optimization algorithms.

What is included in $\mathcal{A}_D$ This class of methods includes many algorithms. The three operations (i-iii) even allow implicit (or proximal) updates, e.g., updates where the point at which the gradient is evaluated is not explicitly known:

$$x_i^{k+1} = x_i^k - \alpha^k \nabla f_i(x_i^{k+1}) \qquad \text{for all } i = 1, \dots, n.$$

Among the primal-based algorithms, $\mathcal{A}_D$ includes, for example, DGD [NO09], DIGing [NOS16], EXTRA [SLWY14], NIDS [LSY17], Acc-DNGD [QL19], Acc-GT [LL21], Mudag [YLZZ23] and their variations [NOSU17, XZSX15, LL20, Jak18]. Among the dual-based algorithms, $\mathcal{A}_D$ includes, for example, the Distributed Dual Averaging [DAW12], MSDA [SBB+17], MSDP [SBB+18], APAPC [KSR20], OPTRA [XTSS20], APM [LFYL20] and others [ULGN20]. The decentralized version of ADMM [BPC+11, SLY+13] does not explicitly use average consensus for the agent interactions, but it can still be reformulated to fit in the class $\mathcal{A}_D$, see e.g., the weighted decentralized version of ADMM [LLST16].

What is NOT included in $\mathcal{A}_D$ The class $\mathcal{A}_D$ is quite general and contains many existing or future methods. However, it does not include adaptive methods (that adjust the step-sizes to the current situation), stochastic methods (that rely on a stochastic gradient oracle), or methods designed for directed communication graphs. Indeed, the latter generally rely on the push-sum consensus protocol [KDG03], which involves quotients of variables that are not permitted in our framework $\mathcal{A}_D$. However, as detailed in Section 2.3.1, this thesis focuses on undirected communications.

2.6 Performance evaluation of decentralized methods

In optimization, the assessment of the performance of a method is generally based on worst-case guarantees. This is also the case in decentralized optimization, where a worst-case guarantee consists in an upper bound on a given performance criterion $\mathcal{P}$ of t iterations of algorithm $A \in \mathcal{A}_D$ with n agents, valid for any starting points satisfying a set of initial conditions $\mathcal{I}$, any local functions in a given set of functions $\mathcal{F}$ and any averaging matrix in a given set of matrices $\mathcal{W}$. Therefore, the guarantee is specific to a performance setting:

$$\mathcal{S}_n = \{n, A, \mathcal{P}, t, \mathcal{I}, \mathcal{F}, \mathcal{W}\}.$$

Accurate guarantees on the worst-case performance of decentralized algorithms are crucial for understanding how their performance is influenced by their parameters, the conditioning of objective functions, and the network topology, thereby enabling correct tuning and comparison of different algorithms. However, deriving such bounds is often challenging, as it requires a proper combination of the impacts of the optimization and communication components. This can lead to highly intricate or conservative bounds, as in Fig. 1.2 in the introduction. Moreover, theoretical analyses often require specific settings that differ from one algorithm to another, making them difficult to compare.

We note that recent results have proposed decentralized algorithms with proven optimal asymptotic complexities, for certain specific performance criteria and classes of functions, see Section 2.4. In particular, most of them focus on time-constant averaging matrices. The performance guarantees for these methods are particularly intricate and may have large hidden constants, preventing them from reaching the optimal rate before a sufficiently large number of iterations, as we will see in Chapter 8. In addition, these methods can be complex to implement in practice, because they use the gradients of local Fenchel conjugates or involve an inner communication loop. Therefore, it remains more than necessary to have access to accurate and comparable guarantees on the worst-case performance of decentralized algorithms, to facilitate and improve their tuning, comparison, and design.

All these reasons motivate the work carried out in this thesis: we develop and exploit tools allowing to automatically and accurately compute

numerical guarantees on the performance of decentralized optimization algorithms. Our new tools rely on a computational approach that finds a worst-case performance guarantee of an algorithm by solving an optimization problem, known as the performance estimation problem (PEP), recently introduced in [DT12, Tay17] and which is detailed in Chapter 3.

2.7 Vector notations

Let $x_i^k \in \mathbb{R}^d$ denote the k-th local variable x of agent i. The local variable of all the agents can be stacked vertically in $\mathbf{x}^k \in \mathbb{R}^{nd}$:

$$\mathbf{x}^k = \begin{bmatrix} x_1^k \\ \vdots \\ x_n^k \end{bmatrix},$$

We can also gather different iterates $k = 0, \dots, t$ in a matrix $X \in \mathbb{R}^{nd \times t+1}$:

$$X = \begin{bmatrix} \mathbf{x}^0 & \dots & \mathbf{x}^t \end{bmatrix}.$$

These vector notations apply to any variable that is used in the decentralized algorithms and assume that the local variable exists for each agent and each iteration, but our work also applies if the variables are not defined for all agents or all iterations of the algorithm. Furthermore, our approach also applies if there is no clear concept of iterations. Using these vector notations, we can write the consensus step (2.18) for all agents at once as

$$\mathbf{y}^k = (W^k \otimes I_d)\mathbf{x}^k, \qquad \text{for } k = 0, \dots, t \tag{2.19}$$

where $W^k \in \mathbb{R}^{n \times n}$ is the averaging matrix used in the consensus step k, I_d denotes the identity matrix of size d and $\otimes$ the Kronecker product. This latter notation means that we apply the same matrix W^k for each dimension of the agent variables. Moreover, if the same averaging matrix W is used for different consensus steps ($k = 0, \dots, t$), we can write all the consensus steps at once using matrix notations:

$$Y = (W \otimes I_d)X. \tag{2.20}$$

The agent average of iterate $\mathbf{x}^k \in \mathbb{R}^{nd}$ is denoted $\overline{x}^k \in \mathbb{R}^d$ and is defined as $\overline{x}^k = \frac{1}{n}\sum_{i=1}^n x_i^k$. We can also gather the agent average of different iterates $k = 0, \dots, t$ in a matrix $\overline{X} \in \mathbb{R}^{d \times t+1}$: $\overline{X} = \begin{bmatrix} \overline{x}^0 & \dots & \overline{x}^t \end{bmatrix}$. Finally, we denote the global average over all the iterates as $x_{\text{av}} = \frac{1}{t+1}\sum_{k=0}^t \overline{x}^k$.

Optimization
Always in service of
Optimization.

3

Performance Estimation Problems

A TIGHT worst-case performance bound on an optimization algorithm could in principle be obtained by running it on every single instance - function and initial condition - allowed by the setting considered and selecting the worst performance obtained. This would also directly provide an example of "worst" instance if it exists. The performance estimation problem (PEP) formulates this abstract idea as a real optimization problem that maximizes the error measure of the algorithm result, over all possible functions and initial conditions allowed [DT12]. This optimization problem is inherently infinite-dimensional, as it contains a continuous function among its variables. Nevertheless, Taylor et al. have shown [THG15b, THG15a] that PEP can be solved exactly using a semidefinite program (SDP) reformulation, for a wide class of (centralized) first-order algorithms and different classes of functions. In this chapter, we summarize results from [THG15b, THG15a] to present what are the elements composing a PEP problem, what are the settings for which it can be written as an SDP, and how this reformulation works. Understanding the general PEP theory will allow us to extend and adapt it to decentralized optimization in Part I of the thesis.

3.1 Elements of a PEP

Let us consider a classical optimization problem

$$\min_{x \in \mathbb{R}^d} f(x), \tag{3.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a real convex function with (at least) one real minimizer $x^* \in \mathbb{R}$. The main ingredients of a Performance Estimation Problem for computing a worst-case guarantee for a first-order algorithm A solving (3.1), are

(i) the first-order optimization algorithm A, e.g., the gradient descent $x^{k+1} = x^k - \alpha \nabla f(x^k)$;

(ii) a performance measure $\mathcal{P}$, e.g., $f(x^k) - f(x^*)$;

(iii) a class of functions $\mathcal{F}$, e.g., the class $\mathcal{F}_{\mu,L}$ of μ-strongly convex and L-smooth functions;

(iv) a set $\mathcal{I}$ of initial conditions, e.g., $\|x^0 - x^*\|^2 \leq 1$.

These ingredients can be organized into an optimization problem that computes a tight worst-case performance guarantee w:

$$\begin{aligned} w(\mathcal{S}) = \max_{f, x^0, \ldots, x^t, x^*} \quad & \mathcal{P}\left(f, x^0, \ldots, x^t, x^*\right) \\ \text{s.t.} \quad & f \in \mathcal{F}, \\ & x^1, \ldots, x^t \text{ are iterates from method } A \text{ applied on } f, \\ & x^* \in \operatorname{argmin} f, \\ & x^0 \text{ satisfies } \mathcal{I}, \end{aligned} \tag{3.2}$$

where $\mathcal{S}$ is the performance evaluation setting which specifies the algorithm A, the number of steps t, the performance criterion $\mathcal{P}$, the class of functions $\mathcal{F}$, and the initial conditions $\mathcal{I}$:

$$\mathcal{S} = (A, t, \mathcal{P}, \mathcal{F}, \mathcal{I}).$$

This formulation of the PEP problem is not directly tractable since it contains a continuous function in its decision variables, which makes it infinite-dimensional. In the next sections, we detail how to obtain a finite

formulation of a PEP problem, based on appropriate interpolation conditions, and then how it can be written as a semidefinite program (SDP), which can be solved efficiently.

3.2 Finite formulation and convex interpolation

To render problem (3.2) finite, rather than considering the function f as a whole, we only consider the discrete set $\mathcal{D} = \{(x^k, g^k, f^k)\}_{k\in I=\{0,\dots,t,*\}}$ of iterates x^k together with their gradient-vectors $g^k = \nabla f(x^k)$ and function values $f^k = f(x^k)$. These are the only points of the functions that matter for the PEP (3.2). The only constraint involving the entire function is the function class constraint, $f \in \mathcal{F}$. This constraint can be replaced by interpolation conditions on the discrete set $\mathcal{D}$ to ensure its consistency with an actual function $f \in \mathcal{F}$, in the sense that the set is *$\mathcal{F}$-interpolable*.

Definition 3.1 ($\mathcal{F}$-interpolable [THG15a])**.** Let I be a finite index set and $\mathcal{F}$ be a class of smooth[1] functions, and consider the set of triples $\mathcal{D} = (x^k; g^k; f^k)_{k\in I}$ where $x^k \in \mathbb{R}^d$, $g^k \in \mathbb{R}^d$ and $f^k \in \mathbb{R}$ for all $k \in I$. The set $\mathcal{D}$ is $\mathcal{F}$-interpolable if and only if there exists a function $f \in \mathcal{F}$ such that both $\nabla f(x^k) = g^k$ and $f(x^k) = f^k$ hold for all $k \in I$.

Interpolation conditions, ensuring that the discrete set $\mathcal{D}$ is $\mathcal{F}$-interpolable, are provided for different classes of functions in [THG15a, Section 3] and [Tay17, Chapter 3]. We define the two common classes of functions that are used in this thesis and give their interpolation conditions, for the sake of completeness.

Definition 3.2 ($\mathcal{F}_{\mu,L}$)**.** A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is in the class $\mathcal{F}_{\mu,L}$ if f is L-smooth (Definition 2.3) and μ-strongly convex (Definition 2.4), with $0 \le \mu \le L$.

Definition 3.3 ($\mathcal{F}_R$)**.** A function $f : \mathbb{R}^d \to \mathbb{R}$ is in set $\mathcal{F}_R$ if f is convex and R-Lipschitz (Definition 2.2), i.e., with bounded subgradients.

Proposition 3.4 (Interpolation constraints for $\mathcal{F}_{\mu,L}$ [THG15a, Theorem 3.3.])**.** *Let I be a finite index set and $\mathcal{F}_{\mu,L}$ the set of L-smooth and μ-strongly convex functions with $\mu < L$. A set of triplets $\{(x^k, g^k, f^k)\}_{k\in I}$ is $\mathcal{F}_{\mu,L}$-interpolable*

[1] The definition of $\mathcal{F}$-interpolability also adapts to classes of non-smooth convex functions, using the subgradients of the function, instead of its gradient.

if and only if the following conditions hold for every pair of indices $k \in I$ and $l \in I$:

$$f^k - f^l - (g^l)^T(x^k - x^l) \geq \frac{1}{2(1-\mu/L)}\left(\frac{1}{L}\|g^k - g^l\|^2 + \mu\|x^k - x^l\|^2 - 2\frac{\mu}{L}(g^k - g^l)^T(x^k - x^l)\right).$$

Extension to the case where $\mu = L$ is provided in [RH24].

Proposition 3.5 (Interpolation constraints for $\mathcal{F}_R$ [THG15a, Theorem 3.5.])**.** *Let I be a finite index set and $\mathcal{F}_R$ the set of convex and R-Lipschitz functions. A set of triplets $\{(x^k, g^k, f^k)\}_{k\in I}$ is $\mathcal{F}_R$-interpolable if and only if the following conditions hold for every pair of indices $k \in I$ and $l \in I$:*

$$f^k - f^l - (g^l)^T(x^k - x^l) \geq 0$$
$$\|g^l\|^2 \leq R^2$$

Using appropriate interpolation conditions, the PEP problem (3.2) can be expressed only in terms of the discrete set $\{x^k, g^k, f^k\}_{k\in I=\{0,\dots,t,*\}}$.

$$w(\mathcal{S}) = \max_{\{x^k,g^k,f^k\}_{k\in I=\{0,\dots,t,*\}}} \mathcal{P}\left(\{x^k, g^k, f^k\}_{k\in I}\right) \tag{3.3}$$
$$\text{s.t. } \{x^k, g^k, f^k\}_{k\in I} \text{ satisfies interpolation constraints for } \mathcal{F}$$
$$x^1, \dots, x^t \text{ are iterates from method } A,$$
$$g^* = 0,$$
$$x^0 \text{ satisfies } \mathcal{I}.$$

It has been shown in [THG15a] that the two problems (3.2) and (3.3) are equivalent, provided that the interpolation conditions for $\mathcal{F}$ are necessary and sufficient. In PEP (3.3), the performance criterion, the initial conditions, and the interpolation constraints are usually quadratic and potentially non-convex in the iterates and gradients and linear in the function values. Problem (3.3) is thus a (non-convex) quadratically constrained quadratic problem (QCQP), which is difficult to solve in general. However, we can reformulate the problem into a convex semidefinite program (SDP).

3.3 SDP reformulation

In many settings, all the PEP elements can be written with expressions linear in the scalar products of the iterates and gradients and in the function values. For algorithm constraints, e.g., $x^{k+1} = x^k - \alpha g^k$, and other constraints that are linear in the iterates and gradient vectors, this can be done by placing all terms on the same side and squaring the constraint. Alternatively, linear equality constraints can be used to substitute vectors with combinations of others, thus reducing the problem dimension.

Whatever the case, since all the PEP elements can be written with expressions linear in the scalar products of the iterates and gradients and in the function values, we can consider these scalar products and function values directly as decision variables of the PEP. For this purpose, we define a vector of functions values $\mathbf{f}$ and a Gram matrix G that contains scalar products between all vectors, e.g., the iterates $x^k \in \mathbb{R}^d$ and the gradients vectors $g^k \in \mathbb{R}^d$.

$$\mathbf{f} = [f^0, \dots, f^t, f^*], \tag{3.4}$$

$$G = P^T P, \quad \text{with} \quad P = \left[x^0, \dots, x^t, x^*, g^0, \dots, g^t, g^*\right]. \tag{3.5}$$

By definition, G is symmetric and positive semidefinite. Moreover, every matrix $G \succeq 0$ corresponds to a matrix P whose number of rows is equal to rank G. It has been shown in [THG15a] that the reformulation of a PEP (3.2) with $(\mathbf{f}, G)$ as variables is lossless provided that we do not impose the dimension d of the worst-case, and look for the worst-case over all possible dimensions (imposing the dimension would correspond to adding a typically less tractable rank constraint on G). The resulting SDP PEP formulation takes thus the form of

$$\begin{aligned} w(\mathcal{S}) \quad = \quad & \max_{\mathbf{f}, G} \quad \mathcal{P}(\mathbf{f}, G) \\ & \text{s.t.} \quad G \succeq 0, \\ & \quad \mathbf{f}, G \quad \text{satisfy} \quad \text{interpolation constraints for } \mathcal{F}, \\ & \qquad\qquad\quad\ \ \text{algorithm constraints for } A, \\ & \qquad\qquad\quad\ \ \text{optimality condition for (3.1)}, \\ & \qquad\qquad\quad\ \ \text{initial conditions } \mathcal{I}. \end{aligned} \tag{3.6}$$

Such an SDP formulation is convenient because it can be solved numerically to global optimality, leading to a tight worst-case bound, and it also provides the worst-case solution over all possible problem dimensions. We refer the reader to [THG15a] for more details about the SDP formulation of a PEP. We note that, in practice, the size of matrix G (3.5) can be reduced by considering only the scalar products between vectors that cannot be expressed as linear combinations of the others. However, the dimension of G always depends on the number of iterations t. From a solution $(\mathbf{f}, G)$ of the SDP formulation, we can construct a solution for the discretized variables $\{(x^k, g^k, f^k)\}_{k\in I}$, e.g., using the Cholesky decomposition of G. Since this set of points satisfies the interpolation constraints for $\mathcal{F}$, we can also construct a corresponding worst-case local function from $\mathcal{F}$ interpolating these points.

Proposition 3.7 states sufficient conditions under which a PEP in the form of (3.2) can be formulated as an SDP. These conditions are satisfied in most PEP settings, allowing to formulate and solve the SDP PEP formulation for a large class of (implicit and explicit) first-order methods, with different classes of functions and performance criteria, see [THG15a]. The proposition uses the following definition.

Definition 3.6 (Gram-representable [THG15a])**.** Consider a Gram matrix G and a vector $\mathbf{f}$, as defined in (3.4)-(3.5). We say that a constraint or an objective is linearly (resp. LMI) Gram-representable if it can be expressed using a finite set of linear (resp. Linear Matrix Inequality) constraints involving (part of) G and $\mathbf{f}$.

Proposition 3.7 ([THG15a, Proposition 2.6])**.** *If the interpolation constraints of the class of functions $\mathcal{F}$, the satisfaction of the method A, the performance measure $\mathcal{P}$ and the set of constraints I, which includes the initial conditions, are linearly (or LMI) Gram-representable, then, computing the worst-case for criterion $\mathcal{P}$ of method A after t iterations on objective functions in class $\mathcal{F}$ with constraints $\mathcal{I}$ can be formulated as an SDP, with $G \succeq 0$ and $\mathbf{f}$ as variables.*

Remark 3.8. This proposition remains valid when the objective function is the sum of n sub-functions, each belonging to a class of functions with linearly (or LMI) Gram-representable interpolation constraints.

Remark 3.9. In [THG15a], Definition 3.6 and Proposition 3.7 were only formulated for linearly Gram-representable constraints, but their extension to LMI Gram-representable constraints is direct. Such constraints will be used for extending the PEP framework to decentralized optimization.

3.4 Exploitation of PEP

PEP techniques have allowed answering several important questions in optimization, see e.g., the list in [Tay17], to make important progress in the tuning of certain centralized algorithms including the well-known gradient method (GM), and to design optimal algorithms for different problem settings. This PEP approach has been made widely available *via* a Matlab [THG17] and Python [GMG+22] toolbox, which makes it easy to formulate and solve PEP problems.

3.4.1 Analysis and tuning of methods

The PEP framework has been used to conduct a tight performance analysis of many centralized first-order methods, in various settings, such as the gradient method (and its accelerated version) on smooth strongly convex functions [DT12, THG15b], weakly convex functions [RGP22, RH24], convex functions with a quadratic upper bound [GTD22], smooth strongly convex functions composed with a linear operator [BHG23] and other non-convex settings [GEIGM22, AdKZ23a]. Variants of GM have been analyzed via PEP, including GM with exact line search [DKGT17], proximal GM [THG18], and inexact GM with bounded error on the gradient [DKGT20]. Other centralized first-order algorithms have also been analyzed with PEP approach, in particular, the block coordinate descent method [SL16, AdKZ22, KHG23], the Bregman gradient method (i.e., the mirror descent) [DTdB22], the difference-of-convex algorithm [AdKZ23b], the Alternating Direction Method of Multipliers (ADMM) [ZAdK23], inexact proximal methods [BTB23] and the Chambolle-Pock algorithm [BHG23].

From Proposition 3.7, the SDP formulation of PEP directly applies to the analysis of Gram-representable algorithms. This does not include algorithms with adaptive step-sizes. However, a PEP-based analysis of such algorithms has still been performed in [BTd20] and [DGFST23]. In [BTd20], the authors analyze Polyak steps with an SDP PEP, by using an appropriate change of variables and a parametric analysis. In [DGFST23], the authors analyze nonlinear conjugate gradient methods via a PEP approach, by formulating and solving non-convex QCQPs.

In addition to analyzing optimization problems and methods, the PEP techniques can also be exploited to study worst-case performance of algo-

rithms intended to solve problems involving operators, such as fixed-point problems [RTBG20, Lie21], maximal monotonic inclusion problems [GY20] and variational inequalities problems [GY22, GLG22, GTG22, GTHG23].

3.4.2 Deriving proofs using the Lagrangian dual solutions

Feasible points of a PEP provide lower bounds on the worst-case performance of the given algorithm, whereas feasible points of the Lagrangian dual of a PEP give upper bounds on the algorithm's worst-case performance. Lagrangian dual problems work with a decision variable for each constraint of the primal problem. Therefore, the proof of a worst-case guarantee for an algorithm is obtained by linearly combining the PEP constraints, with coefficients that are the dual variables of the PEP. The proof holds for any feasible solution to the dual problem, and it provides the best possible worst-case guarantee with the optimal dual solution. This PEP feature has been exploited to derive analytic proofs for algorithm performance guarantees, see e.g., [THG18, GDT23]. The main inherent difficulty of this approach is to obtain an analytic expression for the dual PEP solutions. However, it may still be a lot easier than deriving analytical proofs from scratch. This approach has been detailed in [GDT23] and helps to understand general proof structures in first-order optimization. In [GDT23], the authors show how this can be used to generalize performance guarantees onto broader classes of functions or methods, but also to design new optimization methods and derive their corresponding performance guarantees.

3.4.3 Designing optimal methods

The design of an optimal method for settings $\mathcal{S}$ consists in finding the method providing the best worst-case guarantees in these settings, among all the possible methods of a given class, e.g., first-order methods using only past iterates information to compute the new one. Therefore, the design of an optimal method can be seen as a large tuning problem, which searches for the parameters of a very general method minimizing the resulting worst-case performance. PEP can be used to compute the worst-case performance to minimize, resulting in a min-max problem. Such min-max problems are not easy to solve, but the authors in [KF16] rely on the dual PEP to reformulate it as a convex SDP, resulting in the Optimized Gradient Method (OGM), which is optimal for smooth convex

functions, in terms of functional error decrease [KF16, KF17, KF18] and its variant OGM-G, which is optimal for the gradient norm decrease, up to a constant [KF21]. After OGM, much work was devoted to the design of optimal methods using PEP, for different settings, including smooth convex functions [DT16, DT20, PR24], non-smooth convex functions [DT20], smooth strongly convex functions [TD23, DGVPR24], non-smooth composite convex problem [JGR23], smooth non-convex functions and non-smooth weakly convex functions [DGVPR24]. This last work [DGVPR24] does not rely on a convex reformulation of the problem, but formulates it as a non-convex QCQP, and proposes a spatial branch-and-bound algorithm to solve it to global optimality.

3.4.4 Extension and application to decentralized optimization

The PEP framework presented above does not directly apply to decentralized optimization algorithms. Indeed, a worst-case instance in decentralized optimization consists of n worst-case local functions and sequences of iterates (one for each agent), as well as a worst-case communication network. To be able to look for such a worst-case, the classical PEP framework lacks ways of representing the agent interactions that appear in decentralized algorithms. To the best of our knowledge, our work [CH23, CH21, CH22, CH24a] presented in this thesis is the first to adapt the performance estimation framework to decentralized optimization and to exploit it to analyze decentralized methods. For this purpose, we propose necessary interpolation constraints for consensus steps to offer a practical representation of agent interactions in the PEPs, see Chapter 4. This allows identifying the form of the worst-case averaging matrix in many cases. Moreover, further development of such PEP formulations reveals fundamental properties of worst-case performance in decentralized optimization, such as conditions of independence of guarantees in terms of the number of agents and the agent symmetry of the worst-case local functions and variables, see Chapter 6. In addition, it allows us to study how to tune the communication weights in decentralized optimization [CH24b]. Finally, the techniques developed for decentralized optimization PEP have served as inspirations for the analysis of methods with linear operator [BHG23], and it might serve for other PEP problems in the future, such as the efficient PEP analysis of stochastic methods.

3.5 An alternative approach from control theory

Integral Quadratic Constraints (IQC) An alternative approach for automatic performance estimation relies on tools from robust control and is proposed in [LRP14]. They formulate optimization methods as dynamical systems and adapt integral quadratic constraints (IQC) to obtain convergence guarantees. Integral quadratic constraints are a method of stability analysis of dynamical systems in feedback with non-linearities [MR97]. If the dynamical system we analyze is an iterative optimization method, the non-linear component is the gradient of the objective function. The key idea behind IQC is to replace the non-linear component (the gradient) with (necessary) quadratic constraints satisfied by its inputs and outputs. This idea is similar to interpolation constraints in PEP (Section 3.2), but it does not exploit the function values in the constraints, which are therefore generally not sufficient to describe the effect of a gradient. Relying on these quadratic constraints, one can formulate sufficient conditions for the method to converge with rate ρ, by solving an LMI feasibility problem (see [LRP14, Theorem 4]). Then, the best possible rate can be found with a bisection scheme. The feasibility problem relies on finding a valid Lyapunov function (also called a potential function) for which the convergence rate ρ can be guaranteed. One generally looks for a quadratic Lyapunov function of the form:

$$V(\xi^k) = (\xi^k - \xi^*)^T Q (\xi^k - \xi^*),$$

where ξ is a vector with all the state variables of the optimization algorithm, and Q is a positive definite matrix, to be determined in the feasibility problem. One can also consider Lyapunov functions involving a linear term in the function values. The Lyapunov function can be assimilated to the performance criterion in PEP. However, in PEP, the performance criterion is fixed a priori and is not optimized. This IQC-based Lyapunov analysis is well-suited to algorithms and settings that exhibit a geometric (i.e., linear) convergence, such as the smooth and strongly convex setting. In that case, it focuses on one generic iteration of the algorithm to provide a bound on its infinite-horizon convergence rate, by solving a series of small semidefinite programs. In some cases, the technique can also be extended to non-geometric (i.e., sublinear) convergence by considering a lifted state with multiple past iterates variables, and a well-crafted family of Lyapunov functions.

Tight Lyapunov analysis with interpolation conditions Lyapunov analysis of first-order optimization methods can also be applied using interpolation conditions, instead of IQCs, to represent the effect of the gradient in the algorithms, see [TVSL18, VSL23, UBTG24]. Provided that the algorithm is analyzed over a class of functions for which sufficient and necessary interpolation conditions are available, the Lyapunov function obtained is tight in the sense that it guarantees the best worst-case decrease rate possible for any single (or fixed number of) iteration of the algorithm and for any quadratic Lyapunov function. In [TVSL18, UBTG24], the authors detail this tight Lyapunov analysis technique and apply it to discrete-time first-order algorithms. It has also been applied to the analysis of stochastic methods [TB19, VSL21] and continuous-time models [MTB23].

Comparison with PEP Compared to the PEP framework, one of the assets of the tight Lyapunov approach is its tractability, since the SDPs to solve are of small size. On the other hand, it cannot (easily) deal with time-varying steps or sublinear rates, and the PEP framework can achieve better rates by analyzing several steps. Finally, it also appears less amenable to a modular representation since the algorithms need to fit into a given framework of linear time-invariant dynamical systems. The PEP framework allows working directly with a large diversity of methods and is made very modular and user-friendly via the dedicated toolboxes (PESTO [THG17] and PEPit [GMG+22]).

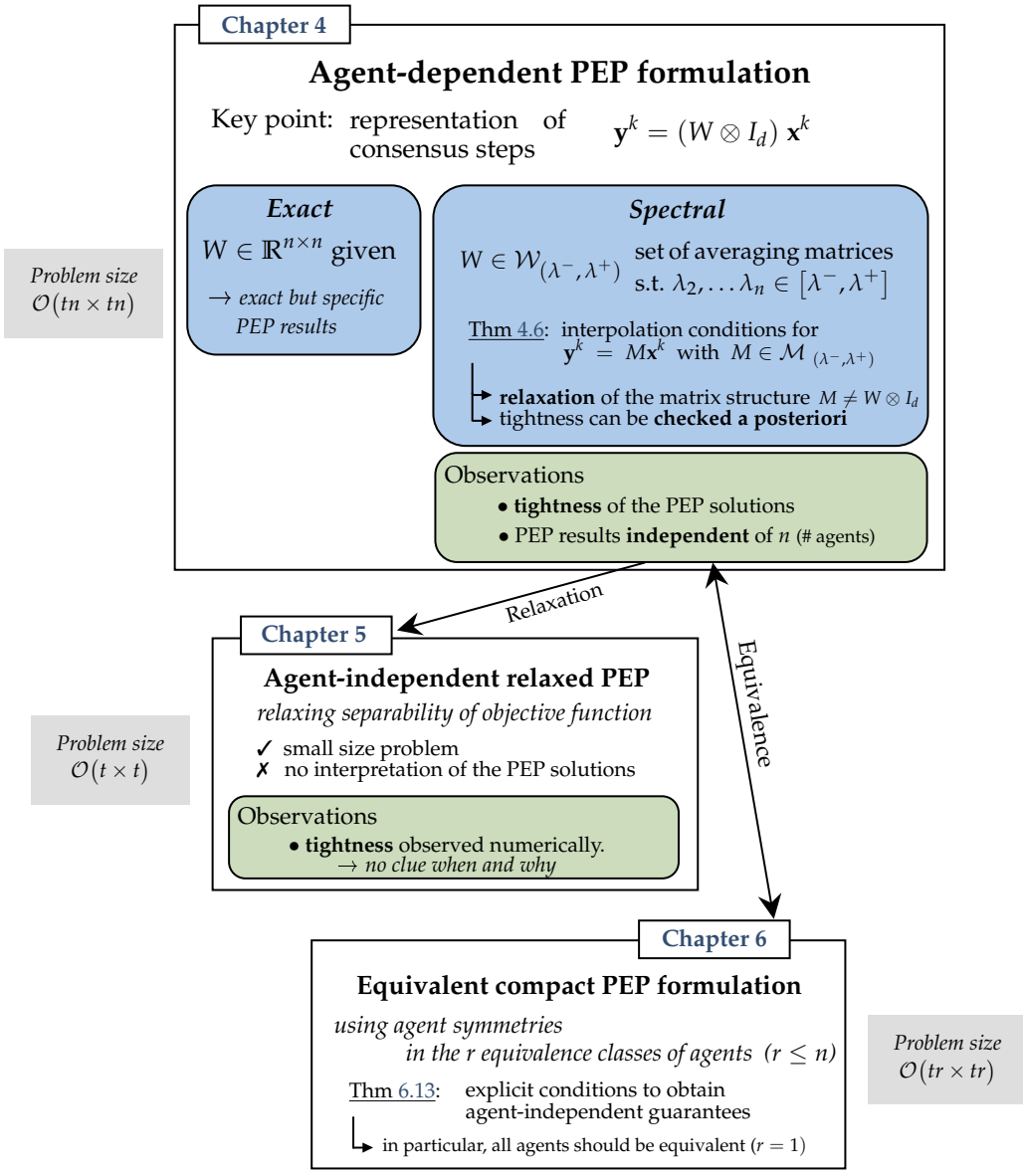
Application to decentralized optimization An extension of the Lyapunov analysis to decentralized optimization is presented in [SVSL20, VSL20] and is also exploited for designing a new decentralized algorithm that achieves a faster worst-case linear convergence rate in the smooth strongly convex setting. Their formulation relies on problems whose size is independent of the number of agents n and it considers a fixed framework of decentralized algorithms that embeds many explicit first-order methods such as DIGing [NOS16], EXTRA [SLWY14] and NIDS [LSY17]. This methodology cannot be directly applied to DGD, nor to smooth convex functions or any other situation that does not have a geometric convergence. Moreover, it can only consider worst-case over time-varying averaging matrices. In comparison, the PEP approach, which is extended to decentralized optimization in this thesis allows in particular the analysis of sublinear rates, time-varying step-sizes, and time-invariant averaging matrices. Moreover,

it applies directly to any decentralized method from $\mathcal{A}_D$, i.e., any decentralized method (implicit or explicit) that involves linear average consensus steps (2.18), without the need to cast the method into a specific framework.

PART I
Performance Estimation Problems for Decentralized Optimization

Visual outline of Part I

The following diagram summarizes the different PEP formulations that we develop in this part of the thesis, in order to analyze the performance of decentralized optimization algorithms over t iterations.



$Ax = \lambda x$
Matrix A transforms the x-vector,
In a linear and elegant fashion.
It's nothing but a factor.
x survives with a small deformation
Because it's a direction of favor,
Bound to lambda, the matrix flesh in true,
Commonly called an eigenvalue.

4

The agent-dependent PEP formulations

We extend the Performance Estimation Problem (PEP) to decentralized optimization, developing an *exact* and a *spectral* agent-dependent PEP formulations which both search for the worst-case local functions and iterates of each agent. The first formulation gives exact worst-case performance but is specific to a given averaging matrix. The second formulation provides worst-case guarantees valid over an entire spectral class of averaging matrices, characterized by the range of their eigenvalues. To build this spectral formulation, we propose necessary interpolation constraints for consensus steps to get a practical representation of agent interactions in PEP. The tightness of this spectral formulation is not a priori guaranteed but can be verified a posteriori, by recovering a valid worst-case averaging matrix.

The chapter is organized as follows:

- Section 4.1 provides a high-level description of the performance estimation problem for decentralized optimization.
- Section 4.2 explains which elements of the decentralized PEP are Gram-representable and can thus be reformulated into a semidefinite program (SDP) with the same techniques as the standard PEP framework from Chapter 3. This includes the consensus steps of decen-

tralized algorithms when the averaging matrix is given. This directly leads to the *exact agent-dependent PEP formulation*.

- Section 4.3 focuses on the representation of consensus steps in PEP, when the averaging matrix can be any matrix from a given spectral matrix class. We provide Gram-representable necessary interpolation constraints to represent such consensus steps and analyze their sufficiency. This leads to the *spectral agent-dependent PEP formulation*, which provides more general worst-case performance guarantees. We also detail how the tightness of this spectral formulation can be verified a posteriori, by recovering a valid worst-case averaging matrix.
- Section 4.4 provides an explicit example of the exact and spectral agent-dependent PEP formulation for the DGD algorithm.
- Section 4.5 presents some numerical results of the spectral PEP formulation to show that its results often appear to be independent of the number of agents. Moreover, we verify the tightness of the results for DGD, by recovering a valid worst-case averaging matrix.

This chapter is based on sections of [CH24a], [CH23] and its preliminary version [CH21].

4.1 High-level PEP description

To compute worst-case performance guarantees for a decentralized optimization algorithm $A \in \mathcal{A}_D$, we can rely on the Performance Estimation Problem (PEP), presented in Chapter 3. One of the main differences in PEP for decentralized optimization is that the problem must find the worst-case for each of the n local functions $\{f_i\}_{i\in\mathcal{V}}$ and the n sequences of local iterates $\{x_i^0 \dots x_i^t\}_{i\in\mathcal{V}}$, with $\mathcal{V} = \{1, \dots, n\}$. The dimension of the problem therefore increases with both the number of iterations t and the number of agents n. A high-level description of a PEP problem for decentralized optimization can be written as follows,

$$
\begin{aligned}
w(\mathcal{S}_n) \;=\; & \max_{x^*,\{x_i^0,\ldots,x_i^t,\, f_i\}_{i\in\mathcal{V}}} \quad \mathcal{P}(f_i, x_i^0, \ldots, x_i^t, x^*) \overset{\text{e.g.,}}{=} \frac{1}{n}\sum_{i=1}^{n} \left(f_i(\bar{x}^t) - f_i(x^*)\right) && (4.1)\\
\text{s.t.} \quad & f_i \in \mathcal{F} \qquad\qquad i \in \mathcal{V}, && \text{(class of functions)}\\
& \{x_i^0, \ldots, x_i^t\}_{i\in\mathcal{V}} \quad \text{from algorithm } A, \ \underset{\text{with } W\in\mathcal{W},}{} && \text{(algorithm)}\\
& \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^*) = 0, && \text{(optimality for (2.1))}\\
& x_i^0 \quad \text{satisfies } \mathcal{I}, \ \underset{\text{e.g., } \|x_i^0 - x^*\|^2 \leq 1,\ i\in\mathcal{V}}{} && \text{(initial conditions)}
\end{aligned}
$$

where $\mathcal{S}_n$ is the performance evaluation setting which specifies the number of agents n, the algorithm A, the number of steps t, the performance criterion $\mathcal{P}$, the class of functions $\mathcal{F}$ for the local function, the initial conditions $\mathcal{I}$ and the class of averaging matrices $\mathcal{W}$ that can be used in the algorithm:

$$\mathcal{S}_n = \{n, A, t, \mathcal{P}, \mathcal{F}, \mathcal{I}, \mathcal{W}\}. \tag{4.2}$$

For simplicity, we assume here that the local functions all belong to the same function class, but the formulation can be directly extended to the case where agents have local functions from different classes.

4.2 SDP reformulation and Gram-representable settings

The same techniques as in Chapter 3 can be applied to discretize problem (4.1) and reformulate it as an SDP. The finite formulation relies on the use of appropriate interpolation conditions on each local function, see Section 3.2. The SDP reformulation of (4.1) relies on the use of $(\mathbf{f}, G)$ as variables; where $\mathbf{f}$ and G are defined similarly as in the centralized case from Section 3.3, but with all agents: $\mathbf{f}$ is the vector of local functions values and G the Gram matrix containing the scalar products between all vectors of the problems, e.g., the local iterates $x_i^k, y_i^k \in \mathbb{R}^d$ and the local gradients vectors $g_i^k \in \mathbb{R}^d$,

$$\mathbf{f} = [f_i^0, \ldots, f_i^t, f_i^*]_{i\in\mathcal{V}} \in \mathbb{R}^{n(t+2)}, \tag{4.3}$$

$$G = P^T P \in \mathbb{R}^{3n(t+2)\times 3n(t+2)}, \quad \text{with } P = \left[y_i^k \; x_i^k \; g_i^k\right]_{i\in\mathcal{V}, k\in I}, \tag{4.4}$$

where $\mathcal{V} = \{1, \ldots, n\}$ and $I = \{0, \ldots, t, *\}$. By definition, G is symmetric and positive semidefinite. As in the centralized case, the size of G can be

reduced in practice, by defining the columns of P as all the vectors used in the performance estimation problem (4.1), that cannot be expressed as linear combinations of the other columns. The resulting SDP PEP problem has the same form as (3.6):

$$\begin{array}{rcll} w(\mathcal{S}_n) & = & \max\limits_{\mathbf{f},G} & \mathcal{P}(\mathbf{f},G) \\ & & \text{s.t.} & G \succeq 0, \\ & & \mathbf{f}, G & \text{satisfy} & \text{interpolation constraints for } \mathcal{F}, \\ & & & & \text{algorithm constraints for } A, \\ & & & & \text{optimality condition for (2.1)}, \\ & & & & \text{initial conditions } \mathcal{I}. \end{array} \tag{4.5}$$

Therefore, according to Proposition 3.7, to be able to write such an SDP PEP formulation with $(\mathbf{f}, G)$ (4.3)-(4.4), we need to use linearly or LMI Gram representable performance criteria, classes of functions, initial conditions and algorithm description (see Definition 3.6). We review below the main settings that can be written linearly with $\mathbf{f}$ and G and are therefore Gram-representable. In particular, we will see that the updates of a decentralized method $A \in \mathcal{A}_D$ with a given averaging matrix are Gram-representable. However, when we allow an entire class of averaging matrices, we do not yet have a Gram representation of the consensus steps, which are part of decentralized algorithms from $\mathcal{A}_D$. Therefore, Section 4.3 below focuses on developing Gram representation of consensus steps with an entire spectral classes of averaging matrices, so that decentralized algorithms can be analyzed in PEP in such settings.

4.2.1 Gram-representable performance criteria

Here is a non-exhaustive list of common performance criteria in decentralized optimization that are linearly Gram-representable and can therefore be used in a PEP. Linear combinations of these expressions also provide linearly Gram-representable performance metrics.

- Agent average of the last local iterate error:

$$\mathcal{P} = \frac{1}{n}\sum_{i=1}^{n} \|x_i^t - x^*\|^2, \tag{4.6}$$

  where x^* is the optimal solution of the decentralized problem 2.1.

- Error of the last agent average iterate:

$$\mathcal{P} = \|\overline{x}^t - x^*\|^2, \qquad \text{where } \overline{x}^t = \frac{1}{n}\sum_{i=1}^{n} x_i^t.$$

- Agent average distance to consensus:

$$\mathcal{P} = \frac{1}{n}\sum_{i=1}^{n} \|x_i^t - \overline{x}^t\|^2.$$

- Average function error at the last iterate:

$$\mathcal{P} = \frac{1}{n}\sum_{i=1}^{n} \left(f_i(x_i^t) - f_i(x^*)\right).$$

- Function error at the last agent average iterate:

$$\mathcal{P} = f(\overline{x}^t) - f(x^*), \tag{4.7}$$

  where $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$.

- Function error at the running average iterate

$$\mathcal{P} = f(x_{\text{av}}) - f(x^*), \tag{4.8}$$

  where $x_{\text{av}} = \frac{1}{t+1}\sum_{k=0}^{t} \overline{x}^k$.

- Squared gradient norm at the last agent average iterate

$$\mathcal{P} = \|\nabla f(\overline{x}^t)\|^2.$$

For each of the above performance criteria, we can verify that it can be expressed linearly in terms of $\mathbf{f}$ and G (4.3)-(4.4).

4.2.2 Gram-representable initial constraints

Any linearly Gram-representable performance criterion $\mathcal{P}$, listed above, can be used or linearly combined in linearly Gram-representable constraints, e.g., $\mathcal{P} \leq D$, with a given constant D. This gives initial constraints when we set $t = 0$ in the above expressions. Other sets of Gram-representable initial constraints include, for example,

- Uniform bound on the initial iterates:

$$\|x_i^0 - x^*\|^2 \leq D^2, \quad \text{for all } i \in \mathcal{V},$$

with a given constant $D \in \mathbb{R}$.

- Uniform bound on the squared norm of the initial local gradients:

$$\|\nabla f_i(x_i^0)\|^2 \leq D^2, \quad \text{for all } i \in \mathcal{V}.$$

- Equal starting points for the agents:

$$x_i^0 = x_j^0, \quad \text{for all } i, j \in \mathcal{V} \qquad \Leftrightarrow \qquad \|x_i^0 - x_j^0\|^2 = 0 \quad \text{for all } i, j \in \mathcal{V}.$$

For each of the above constraint expressions, we can verify that it can be expressed linearly in terms of $\mathbf{f}$ and G (4.3)-(4.4).

4.2.3 Gram-representable classes of functions

There are many classes of functions and operators for which we know linearly Gram-representable interpolations conditions, see [Tay17], [THG15a], [RH24] and the PEPit documentation [GMG+22]. As explained in Chapter 3, the use of necessary and sufficient interpolation conditions is key to formulating a PEP problem in a finite manner. Here are the two Gram-representable classes of functions that we use in this thesis:

- $\mathcal{F}_{\mu,L}$: L-smooth and μ-strongly convex functions with $0 \leq \mu \leq L$, see Definition 3.2. Interpolation conditions for $\mathcal{F}_{\mu,L}$ are given in Proposition 3.4
- $\mathcal{F}_R$: convex and R-Lipschitz functions, see Definition 3.3. Interpolation conditions for $\mathcal{F}_R$ are given in Proposition 3.5.

4.2.4 Gram-representable algorithm constraints

Any decentralized algorithm $A \in \mathcal{A}_D$ with given parameters and coefficients is Gram-representable. Indeed, algorithms from $\mathcal{A}_D$ only combines 3 types of operations (see Definition 2.14):

(i) *Gradients*: each time an agent i samples the gradient of its local function f_i at a point x_i^k, the triplet $\{x_i^k, g_i^k, f_i^k\}$ is added to the set of

points to interpolate to represent the function of agent i, such that $g_i^k = \nabla f_i(x_i^k)$, $f_i^k = f_i(x_i^k)$ and $f_i \in \mathcal{F}$. Therefore, local gradient sampling in the algorithms is handled in PEP via Gram-representable interpolation constraints, previously described.

(ii) *Consensus*: when the averaging matrix $W = [w_{ij}]$ is given[1], the consensus step (2.18) corresponds to a set of linear equality constraints on y_i and x_j, which are equivalent to the constraints

$$\left(y_i - \sum_{j=1}^{n} w_{ij} x_j\right)^T \left(y_i - \sum_{j=1}^{n} w_{ij} x_j\right) = 0, \text{ for } i = 1, \dots, n.$$

These constraints only involve scalar products from G (4.4) and are thus linearly Gram-representable. They can therefore be used in the SDP formulation of a PEP, see Proposition 3.7. Alternatively, linear equality constraints (2.18) can also be used to substitute the values of y_i in the problem, allowing to reduce its size. This solution is also preferable for numerical reasons.

(iii) *Linear Combinations*: Similar reasoning also holds for any linear combination of vectors, with given coefficients.

This allows writing a performance estimation problem providing the exact worst-case performance of a given decentralized method $A \in \mathcal{A}_D$, with specific parameter values and a specific averaging matrix W, given a priori. We call this the *exact agent-dependent PEP formulation*, because the PEP formulation depends on each agent individually. In principle, it can be applied to any matrix W, even those that do not respect the symmetry or the stochasticity assumptions made in Definition 2.7 of averaging matrices. This PEP formulation can be useful for trying different network matrices and observing their impact on the worst-case performance of the algorithm, see Chapter 9.

However, the literature on distributed optimization generally provides bounds on the performance of an algorithm that are valid over a larger set of averaging matrices $\mathcal{W}$, typically based on its spectral properties, see [NOR18] for a survey. Such bounds allow better characterization of the general behavior of the algorithm and can be applied in a wider range of settings. Obtaining such bound with PEP requires having a Gram-

[1] In that case, the class of averaging matrices $\mathcal{W}$, provided in the performance evaluation setting $\mathcal{S}_n$, contains only one matrix, $\mathcal{W} = \{W\}$.

representable representation of the set of all pairs of vectors $\{(\mathbf{x}^k, \mathbf{y}^k)\}$ that can be involved in consensus steps with the same matrix from the given matrix class. The resulting PEP will keep the form of (4.1), with changes only for the decentralized algorithm description, in particular for the consensus steps it involves.

The next section provides necessary constraints for describing the possible pairs of points involved in consensus steps with symmetric and generalized doubly stochastic matrices with a given range on their eigenvalues, and analyzes their sufficiency.

Chronology of results

In [CH23], we introduced a set of necessary constraints to represent consensus steps (2.20) with a spectral class of averaging matrices in PEP. We proved their necessity using classical eigenvalue analysis techniques, but we did not know whether these constraints were sufficient or not, and if not, the precise class it actually described. A few months later, in [BHG23], colleagues Bousselmi et al. established similar constraints to represent the effect of a symmetric linear operator in PEP for centralized optimization. They showed the necessity and sufficiency of their conditions using a constructive proof. Then, in [CH24a], we exploited this result for decentralized optimization, where averaging matrices in consensus steps can be seen as symmetric linear operators, preserving the consensus subspace. The results from [BHG23] allowed us to remove a redundant constraint in the set of constraints from [CH23]. However, in our latest paper [CH24a], we showed that this updated set of constraints is not sufficient to represent consensus steps in PEP because these steps require applying the same averaging matrix to the different vector components. Building on results from [BHG23], we were able to provide a complete characterization of the points allowed by the set of constraints. This thesis presents the most recent and comprehensive results, following the line of thought of our latest paper [CH24a].

4.3 Spectral representation of consensus steps in PEP

In PEP for a decentralized optimization algorithm $A \in \mathcal{A}_D$, with $n \geq 2$ agents, we would like to find Gram-representable constraints to represent all the pairs of points $\{\mathbf{x}^k, \mathbf{y}^k\}$ such that

$$\mathbf{y}^k = (W \otimes I_d)\mathbf{x}^k, \quad \text{for } k = 1, \dots, t, \qquad \text{with } W \in \mathcal{W}_{(\lambda^-, \lambda^+)}, \quad (4.9)$$

where $\mathcal{W}_{(\lambda^-, \lambda^+)}$ is the following set of symmetric averaging matrices, with fixed bounds $\lambda^-, \lambda^+ \in (-1, 1)$ on the eigenvalues (except for $\lambda_1 = 1$):

$$\mathcal{W}_{(\lambda^-, \lambda^+)} = \left\{ W \in \mathbb{R}^{n \times n} : \begin{array}{l} W^T = W \\ \lambda_1(W) = 1,\ v_1(W) = \mathbf{1}/\sqrt{n} \\ \lambda^- \leq \lambda_n(W) \leq \cdots \leq \lambda_2(W) \leq \lambda^+ \end{array} \right\}. \quad (4.10)$$

Such a set is often used to derive theoretical bounds on distributed algorithms [NOR18]. The first two lines in (4.10) are equivalent to conditions (i) and (ii) from Definition 2.7 of an averaging matrix. You may notice that the set $\mathcal{W}_{(\lambda^-, \lambda^+)}$ is not restricted to non-negative matrices, even if the literature often assumes non-negativity through doubly stochasticity. The reasons for this choice are detailed in Remark 2.8 and its impact on the performance guarantees of decentralized algorithms is analyzed in Section 4.5. A set of pairs of points $\{(\mathbf{x}^k, \mathbf{y}^k)\}$ satisfying the consensus steps constraints (4.9) is called $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolable.

Definition 4.1 ($\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolability)**.** Let I be a finite set of indices. A set of pairs $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k \in I}$ is $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolable if,

$$\exists W \in \mathcal{W}_{(\lambda^-, \lambda^+)} \ : \ \mathbf{y}^k = (W \otimes I_d)\mathbf{x}^k \quad \text{for all } k \in I.$$

In this definition, the pairs of points $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k \in I}$ are all linked by the same averaging matrix $W \in \mathcal{W}_{(\lambda^-, \lambda^+)}$. Thus, settings or algorithms using different averaging matrices work with different sets of pairs of points.

This section first shows why there is no Gram-representable constraints to characterize $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolability of a set of pairs of points $\{\mathbf{x}^k, \mathbf{y}^k\}$ involved in consensus steps of the form of (4.9), preventing its tight representation in an SDP PEP formulation. Then we present a relaxed Gram-representable description of $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolable pairs of points and

characterize the points allowed in this relaxed description. We also explain how it can be used to build the *spectral agent-dependent PEP formulation*, which provides worst-case guarantees for a decentralized algorithm, that are valid for any averaging matrix from $\mathcal{W}_{(\lambda^-,\lambda^+)}$. Tightness of the results cannot be guaranteed a priori but can be verified a posteriori by recovering a valid worst-case averaging matrix interpolating the worst-case points, as detailed in Section 4.3.3. Finally, we present some additional theoretical results suggesting that the relaxation of the consensus step representation in PEP should not have a large impact on the PEP results.

4.3.1 The intractable quest for tightness

It would be ideal to find necessary and sufficient conditions for a set of pairs $\{(\mathbf{x}^k, \mathbf{y}^k)\}$ to be $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable, see Definition 4.1. Moreover, the conditions should be Gram-representable, i.e., linear in the scalar products of $\mathbf{x}_i$ and $\mathbf{y}_i$, to be able to use them in an SDP PEP formulation (see Proposition 3.7). In this subsection, we will show, by a non-convexity argument, that there are no tight conditions for $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolability that are Gram-representable.

Theorem 4.2. *There is no Gram-representable necessary and sufficient conditions for $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolability of a set of pairs of variables $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k\in I}$.*

The proof of this theorem relies on a lemma showing the non-convexity of the set of Gram matrices we are considering. To define this set properly, we consider matrices $P_X, P_Y \in \mathbb{R}^{d\times nt}$ aggregating consensus iterates of n agents over t consensus steps.

Definition 4.3 ($\mathcal{G}_{(\lambda^-,\lambda^+)}$)**.** We define $\mathcal{G}_{(\lambda^-,\lambda^+)}$ as the set of symmetric and positive semidefinite Gram matrices of the form

$$G_c = \begin{bmatrix} P_X^T P_X & P_X^T P_Y \\ P_Y^T P_X & P_Y^T P_Y \end{bmatrix}, \quad \text{where} \quad \begin{aligned} P_X &= [\overbrace{x_1^0 \dots x_n^0}^{\mathbf{x}^0} \dots x_1^t \dots x_n^t] \in \mathbb{R}^{d\times nt}, \\ P_Y &= [\underbrace{y_1^0 \dots y_n^0}_{\mathbf{y}^0} \dots y_1^t \dots y_n^t] \in \mathbb{R}^{d\times nt}, \end{aligned}$$

such that the set of pairs $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k=1,\dots,t}$ is $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable, in the sense of Definition 4.1.

In practice, in the PEPs, the Gram matrices can involve other variables than those of the consensus (e.g., gradients). We can therefore consider G_c as a submatrix of the full Gram matrix, which contains only the scalar products related to consensus steps.

Lemma 4.4. *Let $\lambda^- < \lambda^+ \in (-1,1)$. The set $\mathcal{G}_{(\lambda^-,\lambda^+)}$ is non-convex.*

Proof. We build two Gram matrices $G_1, G_2 \in \mathcal{G}_{(\lambda^-,\lambda^+)}$ and show that their combination $G_3 = \frac{1}{2}(G_1 + G_2)$ is not in $\mathcal{G}_{(\lambda^-,\lambda^+)}$. We build a counter-example for one consensus step, i.e., for $t = 1$, and then we explain why it can also be used for $t > 1$. In the set $\mathcal{G}_{(\lambda^-,\lambda^+)}$, there is no constraint on the rank d of the matrices, so we choose to build G_1 and G_2 with rank $d = 1$, as follows

$$\text{let} \qquad \mathbf{x} = \begin{bmatrix} n & 0 & \dots & 0 \end{bmatrix}^T \in \mathbb{R}^n,$$

$$[W_1]_{ij} = \begin{cases} \frac{1+(n-1)\lambda^-}{n} & \text{for } i = j \\ \frac{1-\lambda^-}{n} & \text{for } i \neq j \end{cases}, \quad [W_2]_{ij} = \begin{cases} \frac{1+(n-1)\lambda^+}{n} & \text{for } i = j \\ \frac{1-\lambda^+}{n} & \text{for } i \neq j \end{cases},$$

$$\text{and} \qquad \mathbf{y}_1 = W_1\mathbf{x} = \begin{bmatrix} 1+(n-1)\lambda^- \\ 1-\lambda^- \\ \vdots \\ 1-\lambda^- \end{bmatrix}, \quad \mathbf{y}_2 = W_2\mathbf{x} = \begin{bmatrix} 1+(n-1)\lambda^+ \\ 1-\lambda^+ \\ \vdots \\ 1-\lambda^+ \end{bmatrix}, \tag{4.11}$$

we build G_1 and G_2 as

$$G_1 = \begin{bmatrix} \mathbf{x}\mathbf{x}^T & \mathbf{x}\mathbf{y}_1^T \\ \mathbf{y}_1\mathbf{x}^T & \mathbf{y}_1\mathbf{y}_1^T \end{bmatrix}, \qquad G_2 = \begin{bmatrix} \mathbf{x}\mathbf{x}^T & \mathbf{x}\mathbf{y}_2^T \\ \mathbf{y}_2\mathbf{x}^T & \mathbf{y}_2\mathbf{y}_2^T \end{bmatrix}.$$

The eigenvalues of W_1 are $\{\lambda^-, \dots, \lambda^-, 1\}$, and those of W_2 are $\{\lambda^+, \dots, \lambda^+, 1\}$, so that $W_1, W_2 \in \mathcal{W}_{(\lambda^-,\lambda^+)}$. Therefore, by construction, both pairs $(\mathbf{x}, \mathbf{y}_1)$ and $(\mathbf{x}, \mathbf{y}_2)$ are $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable, and so G_1 and G_2 are in $\mathcal{G}_{(\lambda^-,\lambda^+)}$, and can be combined as $G_3 = \frac{1}{2}(G_1 + G_2)$,

$$G_3 = \begin{bmatrix} \mathbf{x}\mathbf{x}^T & \frac{1}{2}\mathbf{x}(\mathbf{y}_1 + \mathbf{y}_2)^T \\ \frac{1}{2}(\mathbf{y}_1 + \mathbf{y}_2)\mathbf{x}^T & \frac{1}{2}(\mathbf{y}_1\mathbf{y}_1^T + \mathbf{y}_2\mathbf{y}_2^T) \end{bmatrix} = \begin{bmatrix} P_{X_3}^T P_{X_3} & P_{X_3}^T P_{Y_3} \\ P_{Y_3}^T P_{X_3} & P_{Y_3}^T P_{Y_3} \end{bmatrix}. \tag{4.12}$$

We show that, for any dimension $d_3 > 0$, there is no $P_{X_3}, P_{Y_3} \in \mathbb{R}^{d_3 \times n}$ satisfying (4.12) and such that $G_3 \in \mathcal{G}_{(\lambda^-,\lambda^+)}$. Let us proceed by contradiction. Suppose there is some P_{X_3}, P_{Y_3} satisfying (4.12) and

$$P_{Y_3}^T = W_3 P_{X_3}^T, \quad \text{for } W_3 \in \mathcal{W}_{(\lambda^-,\lambda^+)}. \tag{4.13}$$

Then, the block $P_{X_3}^T P_{X_3} = \mathbf{x}\mathbf{x}^T$ has rank one, because $\mathbf{x} \in \mathbb{R}^n$, and thus the block $P_{Y_3}^T P_{Y_3}$, which can be expressed as $P_{Y_3}^T P_{Y_3} = W_3 P_{X_3}^T P_{X_3} W_3$ using

(4.13), also has rank one[2]. However, the block is also defined as $P_{Y_3}^T P_{Y_3} = \frac{1}{2}(\mathbf{y}_1\mathbf{y}_1^T + \mathbf{y}_2\mathbf{y}_2^T)$ from (4.12), which has rank 2 for any $\lambda^- \neq \lambda^+ \in (-1, 1)$, because vectors $\mathbf{y}_1$ and $\mathbf{y}_2$ (4.11) are linearly independent in that case. By this contradiction argument, we have proved that $G_3 = \frac{1}{2}(G_1 + G_2)$ is not in $\mathcal{G}_{(\lambda^-,\lambda^+)}$, meaning that this set is not convex for $t = 1$.

When we consider multiple consensus steps using the same matrix, i.e., when $t > 1$, the counter-example developed above for $t = 1$ can still be chosen for one of the steps, and the Gram matrices G_1, G_2, and G_3 would then be submatrices of the Gram matrices in $\mathcal{G}_{(\lambda^-,\lambda^+)}$, showing the non-convexity of $\mathcal{G}_{(\lambda^-,\lambda^+)}$ for $t > 1$ as well. □

Proof of Theorem 4.2. By definition, Gram-representable constraints defines a convex subset of symmetric and semidefinite Gram-matrices. However, by Lemma 4.4, we know that set $\mathcal{G}_{(\lambda^-,\lambda^+)}$ of Gram-matrices such that the pairs of points $\{\mathbf{x}^k, \mathbf{y}^k\}$ are $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable is a non-convex. Therefore, we know that there exists no Gram-representable constraints that tightly describe all the points $\{\mathbf{x}^k, \mathbf{y}^k\}$ that are $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable. □

4.3.2 A relaxed formulation

We present Gram-representable necessary constraints to represent the set of pairs of points $\{\mathbf{x}^k, \mathbf{y}^k\}$ involved in consensus steps of the form (4.9), i.e., points that are $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable. These constraints describe a convex set of Gram matrices that contains the non-convex set $\mathcal{G}_{(\lambda^-,\lambda^+)}$, and can therefore be exploited in PEP to derive upper bounds on worst-case performance of distributed algorithms. To build this relaxation, we consider a larger class of steps, containing the consensus step as a particular case and derive tight interpolation constraint for it.

Generalized consensus steps The consensus steps (4.9) can be seen as a linear operation on vectors $\mathbf{x}^k \in \mathbb{R}^{nd}$, with a matrix $M \in \mathbb{R}^{nd\times nd}$ that has a specific Kronecker structure:

$$\mathbf{y}^k = M\mathbf{x}^k, \quad \text{for } k = 1, \dots, t \qquad \text{with } M = W \otimes I_d \text{ and } W \in \mathcal{W}_{(\lambda^-,\lambda^+)}.$$

We consider generalized consensus steps where the Kronecker structure is relaxed, so that the vectors $\mathbf{x}^k$ are multiplied by a full matrix with similar

[2] This is due to $\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$.

properties:

$$\mathbf{y}^k = M\mathbf{x}^k, \quad \text{for } k = 1, \dots, t, \qquad \text{with } M \in \mathcal{M}_{(\lambda^-, \lambda^+)}, \tag{4.14}$$

where $\mathcal{M}_{(\lambda^-, \lambda^+)}$ is a set of symmetric matrices that have the same properties as $W \otimes I_d$ with $W \in \mathcal{W}_{(\lambda^-, \lambda^+)}$, but without constraint on their structure. In particular, matrices in $\mathcal{M}_{(\lambda^-, \lambda^+)}$ are still symmetric, generalized doubly stochastic and with the same range of eigenvalues. The set $\mathcal{M}_{(\lambda^-, \lambda^+)}$ is formally defined as follows, for a given range of eigenvalue $\lambda^-, \lambda^+ \in (-1, 1)$, $\lambda^- \leq \lambda^+$:

$$\mathcal{M}_{(\lambda^-, \lambda^+)} = \left\{ M \in \mathbb{R}^{nd \times nd} : \begin{array}{l} M^T = M \\ \lambda_1(M) = \cdots = \lambda_{|\mathcal{C}|}(M) = 1, \\ \text{with eigenvectors forming a basis of } \mathcal{C} \\ \lambda^- \leq \lambda_{nd}(M) \leq \cdots \leq \lambda_{|\mathcal{C}|+1}(M) \leq \lambda^+ \end{array} \right\},$$

where $\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^{nd} \mid x_1 = \cdots = x_n \in \mathbb{R}^d\}$ is the consensus subspace (Definition 2.6) and has dimension $|\mathcal{C}| = d$.

We say that a matrix $M \in \mathcal{M}_{(\lambda^-, \lambda^+)}$ preserves the consensus subspace $\mathcal{C}$ as we have $M\mathbf{v} = \mathbf{v}$ for any $\mathbf{v} \in \mathcal{C}$. Relaxation of the Kronecker structure in a consensus step allows different weights to be applied to each component of the agent vectors, and different components to be mixed.

Interpolation constraints We provide tight interpolation constraints on a set of pair of points $\{\mathbf{x}^k, \mathbf{y}^k\}$ for such generalized consensus steps (4.14), hence providing necessary constraints for describing classical consensus steps (4.9). Interpolation constraints are necessary and sufficient conditions for a set of pair of points $\{\mathbf{x}^k, \mathbf{y}^k\}$ to be interpolable by a matrix $M \in \mathcal{M}_{(\lambda^-, \lambda^+)}$.

Definition 4.5 ($\mathcal{M}_{(\lambda^-, \lambda^+)}$-interpolability)**.** Let I be a finite set of indices. A set of pairs of points $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k \in I}$ is $\mathcal{M}_{(\lambda^-, \lambda^+)}$-interpolable if,

$$\exists M \in \mathcal{M}_{(\lambda^-, \lambda^+)} \ : \ \mathbf{y}^k = M\mathbf{x}^k \quad \text{for all } k \in I.$$

This definition is related to the interpolation of a symmetric linear operator, developed in [BHG23]. In this work, we focus on linear operators preserving a given subspace, namely the consensus subspace $\mathcal{C}$. Based on results from [CH23] and [BHG23], we give necessary and sufficient conditions to have $\mathcal{M}_{(\lambda^-, \lambda^+)}$-interpolability of a set of pairs of points. Before

stating these interpolation conditions, we first introduce a few notions. We aggregate all the vectors $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k\in I}$ in matrices $X, Y \in \mathbb{R}^{nd\times t}$:

$$X = [\mathbf{x}^1 \dots \mathbf{x}^t] \quad \text{and} \quad Y = [\mathbf{y}^1 \dots \mathbf{y}^t]$$

We also consider the decomposition of matrices X and Y in two orthogonal terms:

$$X = X_{//} + X_\perp \text{ and } Y = Y_{//} + Y_\perp,$$

where $X_{//}, Y_{//} \in \mathbb{R}^{nd\times t}$ and $Y_\perp, X_\perp \in \mathbb{R}^{nd\times t}$ are obtained by projecting each column of X and Y respectively on the consensus subspace $\mathcal{C}$ and on its orthogonal subspace $\mathcal{C}^\perp$. This decomposition allows decoupling the effect of the generalized consensus steps (4.14) as follows:

$$Y = MX = M(X_{//} + X_\perp) = X_{//} + MX_\perp \quad \Leftrightarrow \quad Y_{//} = X_{//} \text{ and } Y_\perp = MX_\perp,$$

where $MX_{//} = X_{//}$ because $X_{//} \in \mathcal{C}$ and $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$.

Theorem 4.6 ($\mathcal{M}_{(\lambda^-,\lambda^+)}$ interpolation constraints)**.** *Let I be a set of t indices and $\lambda^-, \lambda^+ \in (-1, 1)$, $\lambda^- \leq \lambda^+$. A set of pairs of points $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k\in I}$ is $\mathcal{M}_{(\lambda^-,\lambda^+)}$-interpolable if, and only if,*

$$X_{//} = Y_{//}, \tag{4.15}$$

$$(Y_\perp - \lambda^- X_\perp)^T (Y_\perp - \lambda^+ X_\perp) \preceq 0, \tag{4.16}$$

$$X_\perp^T Y_\perp = Y_\perp^T X_\perp, \tag{4.17}$$

where $X_{//}, Y_{//} \in \mathbb{R}^{nd\times t}$ contains columns of X and Y projected onto the subspace $\mathcal{C}$, and $X_\perp, Y_\perp \in \mathbb{R}^{nd\times t}$ the columns projected on its orthogonal complement $\mathcal{C}^\perp$.

Remark 4.7. Theorem 4.6 is stated for the case where symmetric linear operators from $\mathcal{M}_{(\lambda^-,\lambda^+)}$ preserve the consensus subspace $\mathcal{C}$ (Definition 2.6), but it actually holds for any general preserved subspace $\mathcal{B} \subseteq \mathbb{R}^{nd}$. Indeed, the proof below remains valid if we replace the consensus subspace $\mathcal{C}$ by any subspace $\mathcal{B}$.

Proof of Theorem 4.6. We need to prove that conditions (4.15), (4.16) and (4.17) are necessary and sufficient for the existence of a matrix $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$ such that $Y = MX$. The proof relies on the use of [BHG23, Theorem 3.3] which proved the necessity and sufficiency of interpolation constraints similar to (4.16) and (4.17), for general linear operators, without any preserved subspace. In our case, we know that the matrix M has no impact on the consensus subspace $\mathcal{C}$ and only acts on the part in $\mathcal{C}^\perp$. We

thus need to use [BHG23, Theorem 3.3] on this part only. Decoupling both parts can be done using an appropriate change of variables:

$$\tilde{X} = \begin{bmatrix} \tilde{X}_{/\!/} \\ \tilde{X}_{\perp} \end{bmatrix} = \begin{bmatrix} Q_{/\!/}^T \\ Q_{\perp}^T \end{bmatrix} X = Q^T X, \tag{4.18}$$

where $Q = [Q_{/\!/}\ Q_{\perp}] \in \mathbb{R}^{nd \times nd}$ is an orthogonal matrix of change of variables and is split into two sets of columns: $Q_{/\!/} \in \mathbb{R}^{nd \times |\mathcal{C}|}$ form a basis of the consensus subspace $\mathcal{C}$ and $Q_{\perp} \in \mathbb{R}^{nd \times |\mathcal{C}^{\perp}|}$ a basis of its orthogonal complement $\mathcal{C}^{\perp}$. The new variable $\tilde{X}$ has thus two sets of components, $\tilde{X}_{/\!/} \in \mathbb{R}^{|\mathcal{C}| \times t}$ describing the coordinates along the subspace $\mathcal{C}$, and $\tilde{X}_{\perp} \in \mathbb{R}^{|\mathcal{C}^{\perp}| \times t}$ describing the coordinates along the orthogonal complement $\mathcal{C}^{\perp}$. By definition, $X_{/\!/}$ and $Y_{/\!/}$ have components only along $\mathcal{C}$ and $X_{\perp}$ and $Y_{\perp}$ only along $\mathcal{C}^{\perp}$:

$$\begin{aligned} X_{/\!/} &= Q \begin{bmatrix} \tilde{X}_{/\!/} \\ 0 \end{bmatrix}, & Y_{/\!/} &= Q \begin{bmatrix} \tilde{Y}_{/\!/} \\ 0 \end{bmatrix}, \\ X_{\perp} &= Q \begin{bmatrix} 0 \\ \tilde{X}_{\perp} \end{bmatrix}, & Y_{\perp} &= Q \begin{bmatrix} 0 \\ \tilde{Y}_{\perp} \end{bmatrix}. \end{aligned} \tag{4.19}$$

By applying this change of variable to condition (4.15), we obtain $\tilde{X}_{/\!/} = \tilde{Y}_{/\!/}$. The two conditions are well equivalent because the change of variable is reversible. By applying this change of variable (4.19) to conditions (4.16) and (4.17), since Q is orthogonal, i.e., $Q^T Q = I$, we obtain

$$\begin{aligned} (\tilde{Y}_{\perp} - \lambda^- \tilde{X}_{\perp})^T (\tilde{Y}_{\perp} - \lambda^+ \tilde{X}_{\perp}) &\preceq 0, \\ \tilde{X}_{\perp}^T \tilde{Y}_{\perp} &= \tilde{Y}_{\perp}^T \tilde{X}_{\perp}. \end{aligned}$$

As the change of variable is reversible, the two sets of constraints are equivalent. By [BHG23, Theorem 3.3], these two constraints are necessary and sufficient for the existence of a symmetric matrix $\tilde{M}_{\perp} \in \mathbb{R}^{|\mathcal{C}^{\perp}| \times |\mathcal{C}^{\perp}|}$ with eigenvalues between λ^- and λ^+ such that $\tilde{Y}_{\perp} = \tilde{M}_{\perp} \tilde{X}_{\perp}$. Therefore, we have proved that conditions (4.15), (4.16) and (4.17) are equivalent to

$$\begin{bmatrix} \tilde{Y}_{/\!/} \\ \tilde{Y}_{\perp} \end{bmatrix} = \begin{bmatrix} I_d & 0 \\ 0 & \tilde{M}_{\perp} \end{bmatrix} \begin{bmatrix} \tilde{X}_{/\!/} \\ \tilde{X}_{\perp} \end{bmatrix} \quad \text{with } \tilde{M}_{\perp} = \tilde{M}_{\perp}^T, \ \ \lambda(\tilde{M}_{\perp}) \in [\lambda^-, \lambda^+]. \tag{4.20}$$

We now need to come back to the initial variables X and Y and characterize the matrix M in terms of $\tilde{M}_{\perp}$. We can use the (reversible) change of variable

(4.18) to express equation (4.20) in terms of Y and X

$$Y = \underbrace{Q \begin{bmatrix} I_d & 0 \\ 0 & \tilde{M}_\perp \end{bmatrix} Q^T}_{M} X \qquad \text{with } \lambda(\tilde{M}_\perp) \in [\lambda^-, \lambda^+].$$

By splitting Q as $Q = [Q_{//}\, Q_\perp]$, matrix M can be written as

$$M = Q_{//}Q_{//}^T + Q_\perp \tilde{M}_\perp Q_\perp^T \qquad \text{with } \lambda(\tilde{M}_\perp) \in [\lambda^-, \lambda^+].$$

Since $\tilde{M}_\perp$ is symmetric, this expression shows that M is also symmetric. We now show that symmetric matrix M has $|\mathcal{C}|$ eigenvalues equal to 1, with associated eigenvectors $Q_{//}$. Let $v_{//}$ denote any column of $Q_{//}$, i.e., any basis vector of the consensus subspace $\mathcal{C}$, we have well that

$$Mv_{//} = Q_{//}Q_{//}^T v_{//} + Q_\perp \tilde{M}_\perp Q_\perp^T v_{//} = v_{//}.$$

Indeed, $Q_\perp^T v_{//} = 0$ and $Q_{//}Q_{//}^T v_{//} = v_{//}$, since $v_{//}$ is a column of $Q_{//}$ which satisfies $Q_\perp^T Q_{//} = 0$ and $Q_{//}^T Q_{//} = I$.
The other eigenvalues of M are equal to the ones of $\tilde{M}_\perp$. Indeed, Let $(v_\perp, \lambda_\perp)$ be a pair of eigenvector and eigenvalue of matrix $\tilde{M}_\perp \in \mathbb{R}^{\mathcal{C}^\perp \times \mathcal{C}^\perp}$, then $(Q_\perp v_\perp, \lambda_\perp)$ is a pair of eigenvector and eigenvalue of matrix $M \in \mathbb{R}^{nd \times nd}$. This can be verified easily as

$$M(Q_\perp v_\perp) = Q_{//}Q_{//}^T(Q_\perp v_\perp) + Q_\perp \tilde{M}_\perp Q_\perp^T (Q_\perp v_\perp) = \lambda_\perp Q_\perp v_\perp,$$

since $Q_{//}^T Q_\perp = 0$ and $Q_\perp^T Q_\perp = I$. □

In Theorem 4.6, we have proved that conditions (4.15), (4.16) and (4.17) are necessary and sufficient for the existence of a matrix $M \in \mathcal{M}_{(\lambda^-, \lambda^+)}$ such that $Y = MX$. We now give some interpretation of the constraints, explain why they are only necessary for $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolability, and describe the impacts in the PEPs for decentralized optimization.

Interpretation of the constraints By Definition 2.6 of the consensus subspace, the orthogonal decomposition terms can be expressed using the agent averages:

$$X_{//} = \overline{X} \otimes \mathbf{1}_n \qquad X_\perp = X - X_{//}, \tag{4.21}$$
$$Y_{//} = \overline{Y} \otimes \mathbf{1}_n \qquad Y_\perp = X - Y_{//}, \tag{4.22}$$

where $\overline{X}$, $\overline{Y} \in \mathbb{R}^{d\times t}$ are agent average of X and Y, i.e., $\overline{X}_{\cdot,k} = \frac{1}{n}\sum_{i=1}^{n} x_i^k$. Therefore, constraint (4.15) is equivalent to

$$\overline{X} = \overline{Y},$$

This constraint is related to the 1 eigenvalues of the matrix and their corresponding eigenvectors, which form a basis for the consensus space $\mathcal{C}$. The constraint imposes that each vector $\mathbf{x}^k$ have the same agents average as $\mathbf{y}^k$, for each generalized consensus step (4.14) ($k = 1, \dots, t$).

Linear matrix inequality (LMI) constraint (4.16) implies, in particular, scalar constraints for the diagonal elements. They correspond to independent constraints for each consensus step, i.e., for each column $x_\perp \in \mathbb{R}^{nd}$ and $y_\perp \in \mathbb{R}^{nd}$ of matrices $X_\perp$, $Y_\perp \in \mathbb{R}^{nd\times t}$:

$$(y_\perp - \lambda^- x_\perp)^T (y_\perp - \lambda^+ x_\perp) \leq 0.$$

This constraint implies in particular that

$$y_\perp^T y_\perp \leq \lambda^2\, x_\perp^T x_\perp, \quad \text{where } \lambda = \max(|\lambda^-|, |\lambda^+|),$$

meaning that the disagreement between the agents, measured by $y_\perp^T y_\perp$ for y and $x_\perp^T x_\perp$ for x, is reduced by a factor $\lambda^2 \in [0, 1)$ after a consensus.

Constraints (4.16) and (4.17) also allow linking different consensus steps to each other, via the impact of off-diagonal terms which exploit the fact that these steps use the same averaging matrix. Moreover, we can interpret constraints (4.16) and (4.17) as a sum over all the dimensions $j = 1, \dots, d$ of the local vectors. Each term of this sum corresponds to the same constraint expression as (4.16) and (4.17), but applied only to matrices $X_\perp^j$ and $Y_\perp^j \in \mathbb{R}^{n\times t}$, containing the j-th component of each vector. Indeed, any product involving $X_\perp$ or $Y_\perp \in \mathbb{R}^{nd\times t}$ can be written as a sum over the dimensions d. For instance, for $X_\perp^T X_\perp$, we have

$$X_\perp^T X_\perp = \sum_{j=1}^{d} (X_\perp^j)^T X_\perp^j.$$

Necessary interpolation conditions for $\mathcal{W}_{(\lambda^-,\lambda^+)}$ In Theorem 4.6, we have proved that conditions (4.15), (4.16) and (4.17) are necessary and sufficient for the existence of a matrix $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$ such that $Y = MX$. By definition of $\mathcal{M}_{(\lambda^-,\lambda^+)}$ and $\mathcal{W}_{(\lambda^-,\lambda^+)}$, these conditions are necessary for the existence of an averaging matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$ such that $Y = (W \otimes I_d)X$, as shown in the following corollary.

Corollary 4.8. *Let* $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k\in I}$ *be a set of* t *pairs of points and* $\lambda^-, \lambda^+ \in (-1,1)$, $\lambda^- \leq \lambda^+$. *If* $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k\in I}$ *is* $\mathcal{W}_{(\lambda^-,\lambda^+)}$*-interpolable (Definition 4.1), then we have*

$$\overline{X} = \overline{Y}, \tag{4.23}$$

$$(Y_\perp - \lambda^- X_\perp)^T (Y_\perp - \lambda^+ X_\perp) \preceq 0, \tag{4.24}$$

$$X_\perp^T Y_\perp = Y_\perp^T X_\perp, \tag{4.25}$$

with $\overline{X}, \overline{Y} \in \mathbb{R}^{d\times t}$ *and* $X_\perp, Y_\perp \in \mathbb{R}^{nd\times t}$ *defined in* (4.21)-(4.22).

Proof. From a matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$, we can always build a matrix $M = (W \otimes I_d) \in \mathcal{M}_{(\lambda^-,\lambda^+)}$, meaning that any set of pairs of points that is $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable is also $\mathcal{M}_{(\lambda^-,\lambda^+)}$-interpolable. Therefore, the points $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k\in I}$ are $\mathcal{M}_{(\lambda^-,\lambda^+)}$-interpolable and satisfy constraints (4.23), (4.24) and (4.25) by Theorem 4.6. □

The reverse statement does not hold since constraints (4.23), (4.24) and (4.25) guarantee the interpolation by a matrix $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$, that may not have the Kronecker structure required to extract an averaging matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$. The full mask of $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$ allows different weights to be applied to each component of the agent and can also mix different components together, which is not the case for consensus steps (4.9) that impose a sparse mask of the form $(W \otimes I_d)$.

Relaxed Performance Estimations Problems Proposition 4.9 below shows that the constraints from Corollary 4.8 are Gram-representable, and by Proposition 3.7, they can therefore be used in SDP PEP formulations. Since the constraints are only necessary for $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolability, they lead to potentially non-tight performance bounds when exploited in PEP for decentralized optimization.

Proposition 4.9. *The constraints* (4.23), (4.24) *and* (4.25) *from Corollary 4.8 are LMI Gram-representable.*

Proof. Constraint (4.23) is linearly (and thus also LMI) Gram-representable because it can be expressed using only elements of G (4.4), i.e., scalar products between x_i^k and y_j^k,

$$\overline{X} = \overline{Y} \quad \Leftrightarrow \quad \left(\frac{1}{n}\sum_i \left(x_i^k - y_i^k\right)\right)^T \left(\frac{1}{n}\sum_j \left(x_j^k - y_j^k\right)\right) = 0 \text{ for } k = 1, ..., t.$$

Constraints (4.24) and (4.25) are LMI Gram-representable because they are LMIs whose entries can be defined using only the entries of G. For example, the entry k,l of $X_\perp^T X_\perp$ can be expressed as the scalar product of columns k and l of $X_\perp$

$$(X_\perp)_{\cdot k}^T (X_\perp)_{\cdot l} = \sum_{i=1}^n \Big(x_i^k - \frac{1}{n}\sum_j x_j^k\Big)^T \Big(x_i^l - \frac{1}{n}\sum_j x_j^l\Big).$$

□

Using necessary constraints (4.23), (4.24) and (4.25) to represent consensus steps of the form of (4.9) in PEP allows obtaining worst-case performance guarantees for a decentralized optimization algorithm $A \in \mathcal{A}_D$, that are valid for any averaging matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$. We therefore call this formulation the *spectral agent-dependent PEP formulation* and its results the *spectral worst-case*. This is an agent-dependent PEP formulation because the PEP it depends on each agent individually. When different averaging matrices are used for different sets of consensus steps in the algorithm, these constraints from Corollary 4.8 can be applied independently to each set of consensus steps. The bounds obtained with this formulation have a priori no tightness guarantees because it considers larger sets of possible consensus iterates $\{\mathbf{x}^k, \mathbf{y}^k\}$, i.e., those that are $\mathcal{M}_{(\lambda^-,\lambda^+)}$-interpolable. However, this relaxation of the Kronecker structure in consensus steps does not appear to impact the worst-case results, as observed numerically in Section 4.5 below. Some theoretical explanations are also given in Section 4.3.4. Moreover, one can always check the tightness of the PEP solution a posteriori, by checking if the provided solution is $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable. If successful, this check also recovers an instance of the worst matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$. This reasoning is detailed in the next section.

4.3.3 Tightness check and worst-case averaging matrix

Corollary 4.8 provides necessary constraints for describing a set of consensus steps $Y = (W \otimes I_d)X$ that use an unknown averaging matrix W from a spectral class $\mathcal{W}_{(\lambda^-,\lambda^+)}$. Indeed, constraints (4.15), (4.16), and (4.17) from Theorem 4.6 guarantee that there is a matrix $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$ such that $Y = MX$, but we have no guarantees about the specific structure of matrix M and we do not know if there is a matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$ such that $Y = (W \otimes I_d)X$.

Recovering a valid worst-case averaging matrix

From a given SDP PEP solution $(\mathbf{f}, G)$ (4.3)-(4.4) of the spectral formulation with range $[\lambda^-, \lambda^+]$, we can factorize the Gram matrix G to obtain associated values for consensus iterates $X, Y \in \mathbb{R}^{nd \times t}$. To check the tightness of the spectral worst-case PEP solution, we need to verify that X, Y are $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolable, meaning that there is a matrix $W \in \mathcal{W}_{(\lambda^-, \lambda^+)}$ such that $Y = (W \otimes I_d)X$. This can be verified by solving the following worst-case averaging matrix estimation problem and checking if $r^* = 0$:

$$r^* = \min_{\hat{W} \in \mathcal{W}_{(\lambda^-, \lambda^+)}} r(\hat{W}) = \|\tilde{Y} - \hat{W}\tilde{X}\|_F \tag{4.26}$$

where $\|\cdot\|_F$ is the Frobenius norm, i.e., $\|A\|_F^2 = \text{tr}(A^T A) = \sum_i \sum_j a_{ij}^2$, and matrices $\tilde{X}, \tilde{Y} \in \mathbb{R}^{n \times td}$ stack the components of each local vector horizontally and thus reshape matrices $X, Y \in \mathbb{R}^{nd \times t}$. This reshaping is needed for recovering a matrix $\hat{W}$ that is identical for every component and has the appropriate size $(n \times n)$. Using the reshaped matrices, the consensus steps can be written as follows: $\tilde{Y} = W\tilde{X}$. Note that problem (4.26) can easily be solved as an SDP, by formulating the constraint $\hat{W} \in \mathcal{W}_{(\lambda^-, \lambda^+)}$ as follows

$$\hat{W}^T = \hat{W} \text{ and } \hat{W}\mathbf{1} = \mathbf{1} \quad \text{and} \quad \lambda^- I \preceq (\hat{W} - \frac{\mathbf{1}\mathbf{1}^T}{n}) \preceq \lambda^+ I$$

For any $\lambda^-, \lambda^+ \in (-1, 1)$, $\lambda^- \leq \lambda^+$, problem (4.26) always has (at least) one optimal solution $\hat{W}^* \in \mathcal{W}_{(\lambda^-, \lambda^+)}$ which gives the matrix from $\mathcal{W}_{(\lambda^-, \lambda^+)}$ that best links $\tilde{X}$ and $\tilde{Y}$. If its optimal cost $r^* = 0$, then the corresponding matrix $\hat{W}^*$ satisfies $\tilde{Y} = \hat{W}^*\tilde{X}$ and $\hat{W}^* \in \mathcal{W}_{(\lambda^-, \lambda^+)}$, meaning the PEP worst-case solution is $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolable, its worst-case value is tight, and $\hat{W}^*$ is a valid worst-case averaging matrix. However, if $r^* > 0$, then there is no matrix $W \in \mathcal{W}_{(\lambda^-, \lambda^+)}$ such that $\tilde{Y} = W\tilde{X}$ and the PEP solution is not $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolable. Since the PEP solution is not guaranteed to be unique, this does not necessarily mean that the resulting PEP worst-case value is conservative since there may be another PEP solution which is $\mathcal{W}_{(\lambda^-, \lambda^+)}$-interpolable and achieve the same performance value.

Alternatively, problem (4.26) without any constraints is a least-square problem whose solution is cheaper to compute and is given by $\hat{W}_{LS} = \tilde{Y}\tilde{X}^\dagger$, where $\tilde{X}^\dagger$ is the pseudo-inverse of $\tilde{X}$. This estimate benefits from the following property.

Proposition 4.10 (Estimating W with pseudo-inverse). *Let $\hat{W}_{LS} = \tilde{Y}\tilde{X}^\dagger$, where $\tilde{X}^\dagger$ is the pseudo-inverse of $\tilde{X}$. If there is a matrix $W \in \mathbb{R}^{n\times n}$ such that $\tilde{Y} = W\tilde{X}$, then we have that*

$$\hat{W}_{LS}\tilde{X} = W\tilde{X} = \tilde{Y}.$$

If, in addition, $\tilde{X} \in \mathbb{R}^{n\times td}$ has linearly independent rows, then $\hat{W}_{LS} = W$.

Proof. By definition, the pseudo-inverse satisfies $\tilde{X}\tilde{X}^\dagger\tilde{X} = \tilde{X}$. Then, if $\tilde{Y} = W\tilde{X}$, we have that

$$\hat{W}_{LS}\tilde{X} = \tilde{Y}\tilde{X}^\dagger\tilde{X} = W\tilde{X}\tilde{X}^\dagger\tilde{X} = W\tilde{X} = \tilde{Y}.$$

If, in addition, $\tilde{X}$ has linearly independent rows, then its pseudo-inverse can be written as $\tilde{X}^\dagger = \tilde{X}^T(\tilde{X}\tilde{X}^T)^{-1}$ and

$$\hat{W}_{LS} = \tilde{Y}\tilde{X}^T(\tilde{X}\tilde{X}^T)^{-1} = W\tilde{X}\tilde{X}^T(\tilde{X}\tilde{X}^T)^{-1} = W.$$

□

Using this result, we can do the following reasoning (summarized in Fig. 4.1) to know if there exists a matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$ that links $\tilde{X}$ and $\tilde{Y}$ and which one it is. We compute the pseudo-inverse estimate $\hat{W}_{LS} = \tilde{Y}\tilde{X}^\dagger$ and its remainder $r(\hat{W}_{LS}) = \|\tilde{Y} - \hat{W}_{LS}\tilde{X}\|_F$. If the remainder is larger than zero, then we can use the contraposition of Proposition 4.10 to state that there is no matrix W such that $\tilde{Y} = W\tilde{X}$. If the remainder is zero, then we still need to check that the estimate $\hat{W}_{LS}$ is symmetric, generalized doubly stochastic, and has the good range of eigenvalues, i.e., $\hat{W}_{LS} \in \mathcal{W}_{(\lambda^-,\lambda^+)}$. If this is the case, then $\hat{W}_{LS}$ is an optimal solution of problem (4.26) with $r(\hat{W}_{LS}) = 0$. Otherwise, $\hat{W}_{LS}$ is not a feasible solution of problem (4.26), and we should probably solve it; except if the rows of $\tilde{X}$ are linearly independent in which case the second part of Proposition 4.10 applies.

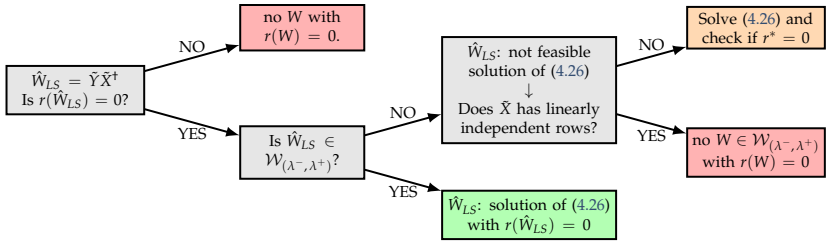


Fig. 4.1 Diagram for existence and estimation of an averaging matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$ such that $r(W) = \|\tilde{Y} - W\tilde{X}\|_F = 0$.

Check tightness using the exact PEP formulation

Let $\mathcal{S}_n^{\text{spectral}} = \{n, A, t, \mathcal{P}, \mathcal{F}, \mathcal{I}, \mathcal{W}_{(\lambda^-,\lambda^+)}\}$ be a performance evaluation setting with a given spectral class of averaging matrices and $\mathcal{S}_n^{\text{fixed}} = \{n, A, t, \mathcal{P}, \mathcal{F}, \mathcal{I}, \{W\}\}$ one with a given and fixed averaging matrix $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$. Let $w(\mathcal{W}_{(\lambda^-,\lambda^+)})$ and $w(W)$ denote the worst-case performance associated with these settings. By definition, since $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$, we have that

$$w(W) \leq w(\mathcal{W}_{(\lambda^-,\lambda^+)}).$$

The lower bound $w(W)$ can be computed using the exact PEP formulation with a fixed averaging matrix, given by W, see Section 4.2. Moreover, an upper bound on $w(\mathcal{W}_{(\lambda^-,\lambda^+)})$ can be computed via the spectral PEP formulation for the set $\mathcal{W}_{(\lambda^-,\lambda^+)}$ of averaging matrices and is denoted by $w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)})$:

$$w(W) \leq w(\mathcal{W}_{(\lambda^-,\lambda^+)}) \leq w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)}). \tag{4.27}$$

If we find an averaging matrix $W^* \in \mathcal{W}_{(\lambda^-,\lambda^+)}$ such that its associated worst-case $w(W^*)$, computed with the exact PEP formulation, is equal to the spectral worst-case $w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)})$, then both bounds in (4.27) are tight and match the worst-case value $w(\mathcal{W}_{(\lambda^-,\lambda^+)})$ that we aim to identify.

$$\begin{aligned} &\text{If} && w(W^*) = w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)}), \\ &\text{then} && w(W^*) = w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)}) = w(\mathcal{W}_{(\lambda^-,\lambda^+)}). \end{aligned}$$

In practice, we can try to find this "worst" averaging matrix W^* by exploring and testing different matrices from $\mathcal{W}_{(\lambda^-,\lambda^+)}$ in the exact PEP formulation. The size of the set $\mathcal{W}_{(\lambda^-,\lambda^+)}$ rapidly increases with the number of agents, making its exploration very time-consuming. To avoid this long exploration, we can try smart guesses for W^*, such as the matrices estimated from the optimal solution of the spectral PEP formulation, i.e., $\hat{W}^*$ computed via (4.26), and measure the gap Δ there is between its specific exact PEP worst-case value $w(\hat{W}^*)$ and the spectral worst-case value $w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)})$. By (4.27), we know that this gap Δ provides an upper bound on the relaxation gap $w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)}) - w(\mathcal{W}_{(\lambda^-,\lambda^+)})$:

$$\Delta = w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)}) - w(\hat{W}^*) \geq w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)}) - w(\mathcal{W}_{(\lambda^-,\lambda^+)}).$$

As explained previously, if $r(\hat{W}^*) = \|\tilde{Y} - \hat{W}^*\tilde{X}\|_F = 0$, then the PEP solution is tight, and we expect to have a zero gap $\Delta = 0$. This test can

serve as a double-check on the tightness of the spectral PEP solution. If $r(\hat{W}^*) > 0$, then we do not know whether the spectral PEP worst-case guarantee $w_{\text{sp}}(\mathcal{W}_{(\lambda^-,\lambda^+)})$ is tight, and we can compute Δ to have an upper bound on the relaxation gap. If $\Delta = 0$, then the spectral bound is tight.

These checks are used to show the tightness of the spectral PEP formulation for the DGD algorithm in Section 4.5 and will be systematically performed in Chapter 8.

4.3.4 Impact of the relaxation

We conclude this section with some additional theoretical results that suggests that the relaxation of the consensus step representation in PEP should not have a significant impact on the PEP results.

The structure of the matrix While the literature for distributed optimization usually assumes consensus steps of the form $\mathbf{y} = (W \otimes I_d)\mathbf{x}$ with $W \in \mathcal{W}_{(\lambda^-,\lambda^+)}$, the existing theoretical performance guarantees also appear to use non-tight description of these consensus steps. Indeed, we did not find any theoretical guarantee that exploits the Kronecker structure $(W \otimes I_d)$ of the consensus steps. In particular, many results in decentralized optimization remain valid for generalized consensus steps $\mathbf{y} = M\mathbf{x}$ with $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$ because the proofs generally rely on the convergence analysis of the consensus step, which is not impacted by the structure of $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$, as shown in Proposition 4.11. This proposition extends classical results of consensus theory, such as Corollary 2.10, to the use of matrices $M \in \mathbb{R}^{nd\times nd}$ that do not have the Kronecker structure of $(W \otimes I_d)$. When $d = 1$, we have $M = W \in \mathbb{R}^{n\times n}$ and no extension is needed.

Proposition 4.11. *Let $M \in \mathcal{M}_{(\lambda^-,\lambda^+)} \subset \mathbb{R}^{nd\times nd}$ with $\lambda^-, \lambda^+ \in (-1, 1)$ and $\mathbf{x}^0 \in \mathbb{R}^{nd}$ a vector stacking the variables $x_i^0 \in \mathbb{R}^d$ of each agent. If the vector $\mathbf{x}^0$ is updated with a series of general matrix steps $\mathbf{x}^{k+1} = M\mathbf{x}^k$, then the sequence of vectors $\mathbf{x}^k$ converges to the agent average consensus vector, i.e., a vector stacking n times the agent average $\overline{x}^0 = \frac{1}{n}\sum_{i=1}^n x_i^0$:*

$$\lim_{k\to\infty} \mathbf{x}^k = \mathbf{1}_n \otimes \overline{x}^0.$$

Proof. Since $M \in \mathcal{M}_{(\lambda^-,\lambda^+)}$, the matrix admits the following spectral decomposition:

$$M = V\Sigma V^T, \qquad \text{with} \qquad V_{1,\dots,d} = (\mathbf{1}_n \otimes I_d)/\sqrt{n}, \tag{4.28}$$

where V is the orthogonal matrix of eigenvectors of M and Σ the diagonal matrix of eigenvalues of M. We define a new variable $\mathbf{z}^k = V^T\mathbf{x}^k$, for which the matrix step $\mathbf{x}^{k+1} = M\mathbf{x}^k$ becomes

$$\mathbf{z}^{k+1} = \Sigma\mathbf{z}^k.$$

The d first eigenvalues in Σ are associated with eigenvectors $V_{1,\dots,d}$ and are equal to 1; all the others are smaller than 1 in absolute value. Therefore,

$$\lim_{k\to\infty}\mathbf{z}^k = \lim_{k\to\infty}\Sigma^k\mathbf{z}^0 = \begin{bmatrix} z_1^0 & \dots & z_d^0 & 0 & \dots & 0\end{bmatrix}^T = \tilde{\mathbf{z}}^0,$$

where vector $\tilde{\mathbf{z}}^0$ contains the d first entries of $\mathbf{z}^0$ and $(n-1)d$ zeros. Thus, the sequence $\mathbf{x}^k$ converges to $V\tilde{\mathbf{z}}^0$ which is the combination of the columns $V_{1,\dots,d}$ with coefficients given by the d first entries of $\mathbf{z}^0$, i.e.,

$$\lim_{k\to\infty}\mathbf{x}^k = \lim_{k\to\infty}V\mathbf{z}^k = V\tilde{\mathbf{z}}^0 = V_{1,\dots,d}\,\mathbf{z}^0_{1,\dots,d}.$$

Finally, by definition of variable $\mathbf{z}$, we have $\mathbf{z}^0 = V^T\mathbf{x}^0$, meaning that the d first entries of $\mathbf{z}^0$ are given by $\mathbf{z}^0_{1,\dots,d} = V^T_{1,\dots,d}\mathbf{x}^0$, and therefore:

$$\lim_{k\to\infty}\mathbf{x}^k = V_{1,\dots,d}V^T_{1,\dots,d}\mathbf{x}^0 = \frac{1}{n}(\mathbf{1}_n\mathbf{1}_n^T\otimes I_d)\mathbf{x}^0 = \mathbf{1}_n\otimes\overline{x}^0,$$

where the second equality holds by definition of $V_{1,\dots,d}$ (4.28). □

Convex hull perspective By Propositions 4.9 and 3.7, the interpolation constraints from Corollary 4.8 can be used in an SDP PEP formulation and thus define a convex set of Gram matrices $\mathcal{H}_{(\lambda^-,\lambda^+)}$, which contains the non-convex set $\mathcal{G}_{(\lambda^-,\lambda^+)}$.

Definition 4.12 ($\mathcal{H}_{(\lambda^-,\lambda^+)}$)**.** We define $\mathcal{H}_{(\lambda^-,\lambda^+)}$ as the convex set of symmetric and positive semidefinite Gram matrices of the form

$$G_c = \begin{bmatrix} P_X^TP_X & P_X^TP_Y \\ P_Y^TP_X & P_Y^TP_Y\end{bmatrix}, \quad \text{where} \quad \begin{aligned} P_X &= \begin{bmatrix} x_1^0 \dots x_n^0 \dots x_1^t \dots x_n^t\end{bmatrix} \in \mathbb{R}^{d\times nt}, \\ P_Y &= \begin{bmatrix} y_1^0 \dots y_n^0 \dots y_1^t \dots y_n^t\end{bmatrix} \in \mathbb{R}^{d\times nt},\end{aligned}$$

such that the points aggregated in matrices $X, Y \in \mathbb{R}^{nd\times t}$ satisfy interpolation constraints (4.23), (4.24) and (4.25) from Corollary 4.8.

When we consider only one consensus step ($t = 1$), we can show that this convex set $\mathcal{H}_{(\lambda^-,\lambda^+)}$ is the smallest convex set containing $\mathcal{G}_{(\lambda^-,\lambda^+)}$, i.e.,

its convex hull.

Theorem 4.13. *Let $t = 1$, meaning we focus only on one consensus step. The set of Gram matrices $\mathcal{H}_{(\lambda^-,\lambda^+)}$, that we describe using interpolation constraints from Corollary 4.8 (see Definition 4.12), is the convex hull of the set of Gram matrices $\mathcal{G}_{(\lambda^-,\lambda^+)}$ that are consistent with a consensus step (see Definition 4.3):*

$$\mathcal{H}_{(\lambda^-,\lambda^+)} = \text{ConvexHull}\{\mathcal{G}_{(\lambda^-,\lambda^+)}\} \qquad \textit{when} \quad t = 1.$$

The proof of Theorem 4.13 relies on the use of Lemma 4.14 which states that the intersection of the SDP cone with a linear constraint always admits a rank one decomposition.

Lemma 4.14 (Corollary 4 from [SZ03])**.** *Let $G \in \mathbb{R}^{n\times n}$ be a positive semidefinite matrix of rank r and $A \in \mathbb{R}^{n\times n}$ be an arbitrary symmetric matrix. If $\text{tr}(A^TG) \geq 0$, then we can always find vectors $p_i \in \mathbb{R}^n$ allowing to write a rank-one decomposition of G such that*

$$G = \sum_{i=1}^{r} p_i p_i^T \quad \textit{and} \quad p_i^T A p_i = \frac{\text{tr}(A^TG)}{r} \geq 0 \quad \textit{for all } i = 1,\dots,r.$$

Proof of Theorem 4.13. To prove that $\mathcal{H}_{(\lambda^-,\lambda^+)}$ is the convex hull of $\mathcal{G}_{(\lambda^-,\lambda^+)}$, we show that all the extreme rays of $\mathcal{H}_{(\lambda^-,\lambda^+)}$ are also in $\mathcal{G}_{(\lambda^-,\lambda^+)}$. A simple case where this holds true is when the extreme rays of $\mathcal{H}_{(\lambda^-,\lambda^+)}$ are rank-one matrices. Indeed, any rank-one Gram matrix from $\mathcal{H}_{(\lambda^-,\lambda^+)}$ is $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable, and hence in $\mathcal{G}_{(\lambda^-,\lambda^+)}$. A rank-one Gram matrix from $\mathcal{H}_{(\lambda^-,\lambda^+)}$ corresponds to one-dimensional variables $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ that are $\mathcal{M}_{(\lambda^-,\lambda^+)}$-interpolable by definition, and hence also $\mathcal{W}_{(\lambda^-,\lambda^+)}$-interpolable, since $\mathcal{M}_{(\lambda^-,\lambda^+)} = \mathcal{W}_{(\lambda^-,\lambda^+)}$ when $d = 1$.

Let us now use Lemma 4.14 to show that the extreme rays of $\mathcal{H}_{(\lambda^-,\lambda^+)}$ are effectively rank-one matrices. By definition, set $\mathcal{H}_{(\lambda^-,\lambda^+)}$ contains SDP matrices G_c that satisfy constraints (4.23), (4.24) and (4.25). When $t = 1$, this set can be described only with the SDP constraint $G_c \succeq 0$ and the scalar constraint (4.24). Indeed, when $t = 1$, constraints (4.24) and (4.25) become scalar constraints, in which case, the constraint of symmetry (4.25) is always satisfied, by symmetry of the scalar product. Moreover, the equality constraint (4.23) does not need to be included since it can be used to express one variable in terms of the others and reduce the number of variables in the Gram matrices. Therefore, for any $G_c \in \mathcal{H}_{(\lambda^-,\lambda^+)}$, Lemma 4.14, applied to $G = G_c \succeq 0$ and matrix A chosen according to constraint (4.24)

with $t = 1$, shows that there exists a rank-one decomposition of G_c. Since all matrices from $\mathcal{H}_{(\lambda^-,\lambda^+)}$ admit a rank-one decomposition, extremes rays of $\mathcal{H}_{(\lambda^-,\lambda^+)}$ are rank-one matrices. □

This result means that if we want to analyze the performance of a distributed algorithm using time-varying consensus steps, we cannot find a better convex description of the effect of each consensus steps than the one of Corollary 4.8. There is currently no proof of such a result for multiple consensus steps using the same averaging matrix, i.e., when $t > 1$.

When facing a non-convex set of Gram matrices in PEP, relaxing this set to its convex hull allows recovering a worst-case solution from the desired non-convex set since the objective function is linear in the Gram matrix G (see [Jag11, Lemma 8]). However, in PEP, we often combine different sets of Gram matrices related to different properties: set of functions, set of averaging matrices, initial conditions, etc. Hence, having an accurate (tight or convex hull) Gram-representable description for each property individually does not guarantee a tight description of their intersection. For the convex hulls, the reason why it arises is explicit:

$$\text{ConvexHull}(\mathcal{G}_1 \cap \mathcal{G}_2) \neq \text{ConvexHull}(\mathcal{G}_1) \cap \text{ConvexHull}(\mathcal{G}_1).$$

The issue of obtaining an exact representation for an intersection of tight sets also arises when analyzing classes of functions or objects combining several properties, e.g., convexity and smoothness [RBCH23]. As explained in [RBCH23], the tight description of smooth convex functions differs from the juxtaposition of the tight descriptions of convex and smooth functions. In general, to conserve tightness, a new ad-hoc description needs to be considered but there is as yet no automatic way of building it.

4.4 Example: the exact and spectral PEP formulations for DGD

We consider t iterations of DGD, described in Algorithm 1, applied on n agents and with constant step-size α and a constant averaging matrix W, in order to solve problem (2.1), i.e., minimizing $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$, with x^* as minimizer of f. We formulate the PEPs under the same settings as the theoretical DGD performance guarantee [NOR18] to which we compare them in Figure 1.2 and Chapter 8. Namely, the evaluation settings are:

- The *class of functions* $\mathcal{F}_R$ (Definition 3.3): all local functions are convex with subgradients norm bounded by R, i.e., $f_i, \dots, f_n \in \mathcal{F}_R$,
- *Initial conditions* impose equal starting points and bounded distance to the optimum:
$$\|x_i^0 - x^*\|^2 \leq D^2 \quad \text{and} \quad x_i^0 = x_j^0 \qquad \text{for all } i, j \in \mathcal{V}.$$
- The *performance criterion* is the function error at the running average iterate (4.8):
$$\mathcal{P} = f(x_{\text{av}}) - f(x^*), \quad \text{with } x_{\text{av}} = \frac{1}{t+1} \sum_{k=0}^{t} \overline{x}^k.$$

In such settings, our exact PEP formulation searches for solutions to the following maximization problem, for a given averaging matrix $W \in \mathbb{R}^{n \times n}$:

$$\max_{\substack{x^*, f_i, x_i^0, y_i^0 \dots x_i^t, y_i^t \\ \text{for } i=1,\dots,n}} \quad \frac{1}{n} \sum_{i=1}^{n} (f_i(x_{\text{av}}) - f_i(x^*)) \qquad \text{(DGD}(W)\text{-PEP)}$$

$$\begin{aligned}
\text{s.t.} \quad & f_i \in \mathcal{F}_R && \text{for } i = 1, \dots, n, \\
& \frac{1}{n} \sum_{i=1}^{n} f_i(x^*) = 0, \\
& \|x_i^0 - x^*\|^2 \leq D^2 \quad \text{and} \quad x_i^0 = x_j^0 && \text{for } i, j = 1, \dots, n, \\
\text{DGD} \quad & \begin{cases} y_i^k = \sum_{j=1}^n w_{ij} x_j^k, \\ x_i^{k+1} = y_i^k - \alpha^k \nabla f_i(x_i^k), \end{cases} && \begin{array}{l} \text{for } i = 1, \dots, n, \\ \text{for } k = 0, \dots, t-1. \end{array}
\end{aligned} \tag{4.29}$$

The objective function and the constraints of (DGD(W)-PEP) are all linearly Gram-representable and the problem can then be formulated as an SDP, according to Proposition 3.7. This is referred to as the *exact PEP formulation* because it finds the exact worst-case performance of the algorithm specific to the given matrix W. The *spectral PEP formulation* relaxes the consensus constraints imposed by equation (4.29) and replaces them with the constraints from Corollary 4.8 with chosen values λ^- and λ^+ for the range of eigenvalues. Those are LMI Gram-representable (see Proposition 4.9) and can then be used in the SDP formulation of PEP, according to Proposition 3.7. This formulation provides *spectral worst-cases*, i.e., upper bounds on the worst-case performances of the algorithm, valid for any matrix $W \in \mathcal{W}_{(\lambda^-, \lambda^+)}$.

Time-varying averaging matrices To obtain a spectral PEP formulation allowing time-varying matrices in DGD,

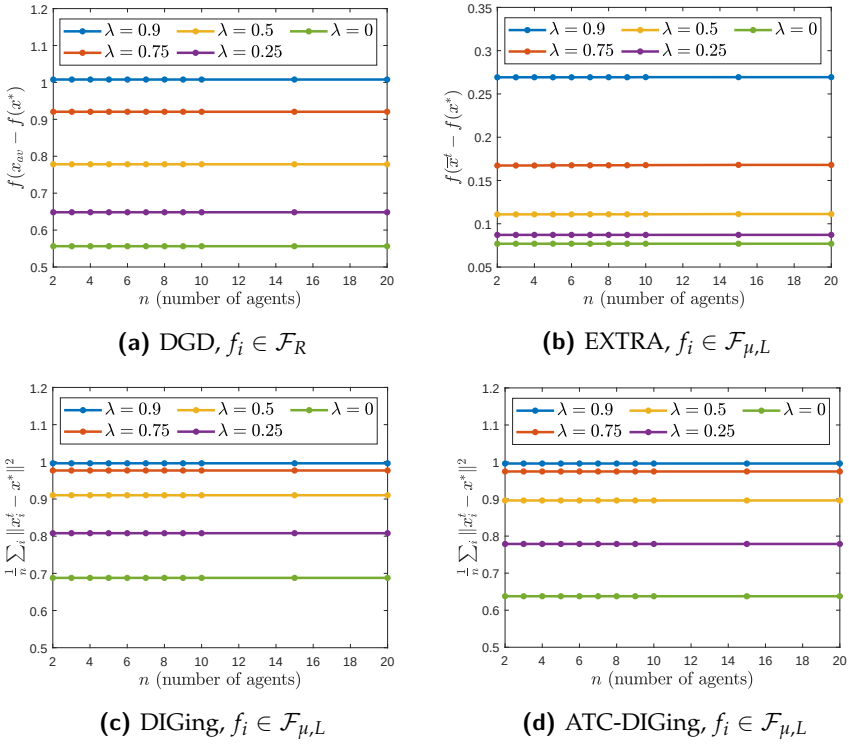
$$y_i^k = \sum_{j=1}^{n} w_{ij}^k x_j^k,$$

we relax these consensus constraints in (DGD(W)-PEP) and replace each of them by the set of constraints from Corollary 4.8 where iterate matrices only contain vector from one iteration k: $X = [\mathbf{x}^k]$, $Y = [\mathbf{y}^k]$. Since X and Y have only one column, the symmetry constraint (4.25) is always satisfied and the LMI constraint (4.24) becomes a scalar inequality.

4.5 Numerical results

The exploitation of our PEP tools for analyzing the performance of decentralized optimization methods is the focus of Chapter 8. Therefore, this section provides only experiments and observations that help to understand and interpret the PEP formulations developed in this chapter and motivate our approach in subsequent chapters. Firstly, we observe that the spectral PEP bounds are independent of the number of agents n in the problem, for a large variety of settings. This observation motivated us to design PEP formulations that are independent of the number of agents and that we present in Chapters 5 and 6. Secondly, we use results from Section 4.3.3 to analyze the tightness of the spectral worst-case bound, including in the case where we restrict our setting to (non-negative) doubly stochastic averaging matrices.

Implementation We have updated the Matlab PESTO toolbox [THG17] to allow easy and intuitive implementation of the exact and spectral PEP formulations for any decentralized optimization methods from $\mathcal{A}_D$. We have also added code examples for a few decentralized algorithms (DGD, DIGing, and Acc-DNGD). We refer to Chapter 7 for more details about the PESTO toolbox and the practical use of the agent-dependent PEP formulations, and to the GitHub repository Performance-Estimation-Problems-for-distributed-optimization for all the code examples with practical implementation of our PEP formulations for distributed optimization.



(a) DGD, $f_i \in \mathcal{F}_R$

(b) EXTRA, $f_i \in \mathcal{F}_{\mu,L}$

(c) DIGing, $f_i \in \mathcal{F}_{\mu,L}$

(d) ATC-DIGing, $f_i \in \mathcal{F}_{\mu,L}$

Fig. 4.2 Independence of the number of agents n for the spectral worst-case performance of $t = 5$ iterations of different algorithms with a constant step-size, a constant averaging matrix from $\mathcal{W}_{(-\lambda,\lambda)}$ and different settings: (a) DGD with $\alpha = 1/\sqrt{t}$ and $f_i \in \mathcal{F}_R$ with $R = 1$, for all $i \in \mathcal{V}$; (b) EXTRA with $\alpha = \frac{1}{2L}$ and $f_i \in \mathcal{F}_{\mu,L}$ with $\mu = 0.1$, $L = 1$, for all $i \in \mathcal{V}$; (c)-(d) DIGing and ATC-DIGing with $\alpha = 0.44(1-\lambda)^2$ and $f_i \in \mathcal{F}_{\mu,L}$ with $\mu = 0.1$, $L = 1$, for all $i \in \mathcal{V}$.

4.5.1 Independence of the number of agents

The size of the PEP formulations for decentralized optimization presented in this chapter is depending on the number of agents in the system, see e.g., (4.5). However, in our numerical experiments, we have observed that the results of the spectral PEP formulation are *independent* of the number of agents $n \geq 2$ in many settings. This is shown in Fig. 4.2 for $t = 5$ iterations of DGD [NO09], EXTRA [SLWY14, LL20], DIGing [NOS16] and ATC-DIGing [NOSU17, XZSX15] and different spectral ranges. We chose to show this independence result for different algorithms, classes of functions, and performance criteria, to show that it seems to hold in a diversity of settings. We chose to use similar criteria than the literature, i.e., $\mathcal{P} = f(x_{\text{av}}) - f(x^*)$ (4.8) for DGD with constant step-size, $\mathcal{P} = f(\overline{x}^t) - f(x^*)$ (4.7) for EXTRA and $\mathcal{P} = \frac{1}{n}\sum_{i=1}^n \|x_i^t - x^*\|^2$ (4.6) for DIGing and ATC-DIGing.

These observations suggest that the spectral worst-case performance could be computed uniformly for all values of n, by solving a PEP problem independent of the number of agents n. In particular, the size of the PEP would be independent of n. This motivates the work done in Chapters 5 and 6. In fact, Chapter 6 exploits the symmetries between agents to obtain an equivalent spectral PEP formulation with a size independent of n, and it also describes the conditions under which the resulting spectral worst-case bounds are guaranteed to be independent of the number of agents n.

4.5.2 Impact of relaxations

In this section, we focus our analysis on the DGD algorithm with a constant step-size, constant averaging matrix, and $n = 3$ agents. This low number of agents allows keeping the computational complexity low while keeping some non-trivial network matrices. As shown in the previous section, the results of the spectral PEP formulation for other numbers of agents are identical.

Worst-case averaging matrix and tightness analysis When considering the spectral PEP formulation for DGD with a symmetric spectral range $-\lambda^- = \lambda^+ = \lambda$, we observe that the worst-case averaging matrix estimation problem (4.26) always has (at least) a solution $\hat{W}^*$ producing a remainder $r(\hat{W}^*) = \|\tilde{Y} - \hat{W}^*\tilde{X}\|_F$ close to zero. The obtained worst-case

averaging matrix has the form

$$\hat{W}^*(\lambda) = (1+\lambda)\frac{\mathbf{1}\mathbf{1}^\mathbf{T}}{n} - \lambda I, \tag{4.30}$$

where I is the identity matrix. This matrix has only two different entries: one for diagonal elements and one for off-diagonal ones,

$$[\hat{W}^*]_{ij} = \begin{cases} \frac{1+\lambda}{n} & \text{if } i \neq j, \\ \frac{1-\lambda(n-1)}{n} & \text{if } i = j. \end{cases}$$

Matrix $\hat{W}^*$ is feasible for (4.26) since it is symmetric and generalized doubly stochastic, leading 1 to be one of its eigenvalues. By definition, all its other eigenvalues are equal to $-\lambda$ and therefore, $\hat{W}^* \in \mathcal{W}_{(-\lambda,\lambda)}$. As explained in Section 4.3.3, a zero remainder $r(\hat{W}^*)$ means that the obtained spectral PEP solution for DGD is $\mathcal{W}_{(-\lambda,\lambda)}$-interpolable and provides a *tight performance bound* for DGD with averaging matrices from $\mathcal{W}_{(-\lambda,\lambda)}$, even though the spectral PEP formulation is a relaxation. The tightness of the spectral PEP formulation for DGD is confirmed by the fact the bound obtained using the exact PEP formulation with this specific matrix $\hat{W}^*$ for $n = 3$, $t = 10$ and plotted in green in Fig. 4.3, matches the spectral bound in blue for all $\lambda \in [0, 1]$. As explained in Section 4.3.3, when the lower and upper bounds on the worst-case performance $w(\mathcal{W}_{(-\lambda,\lambda)})$ coincide, then both bounds are tight. This observation has been confirmed for other values of t (5, 15, and 20).

In Fig. 4.3, we also observe that worst-case performance of DGD is better (i.e., lower) for smaller values of λ. As explained in Section 2.3.2, the value of λ is related to the connectivity of the underlying network and smaller values of λ corresponds to better connected graphs.

Doubly stochastic versus generalized doubly stochastic Since every doubly stochastic matrix is also generalized doubly stochastic, the spectral bound for DGD with averaging matrices from $\mathcal{W}_{(-\lambda,\lambda)}$ also provides an upper bound on the performance of DGD with averaging matrices restricted to (non-negative) doubly stochastic matrices from $\mathcal{W}_{(-\lambda,\lambda)}$. This bound remains tight for $\lambda \leq \frac{1}{n-1}$ because the worst-case matrix $\hat{W}^*$ (4.30) we have obtained is non-negative and is therefore doubly stochastic. For $\lambda > \frac{1}{n-1}$, this is no longer the case and the analysis is performed by empirically looking for symmetric doubly stochastic matrices leading to the worst performance. In Fig. 4.3, for $n = 3$ and $\lambda > 0.5$, we have generated more than 6000 random symmetric doubly stochastic 3 by 3 matrices. We

have analyzed their associated DGD performance using the exact PEP formulation and have only kept those leading to the worst performances. The resulting pink curve deviates no more than 20% below the spectral bound. The curve has three regimes, corresponding to 3 types of worst-case (non-negative) averaging matrices:

$$\begin{cases} \hat{W}^*(\lambda) & \text{for } 0 \leq \lambda < 0.5, \\ \hat{W}^*(0.5) & \text{for } 0.5 \leq \lambda \leq 0.62, \\ \hat{W}^+(\lambda) & \text{for } 0.62 < \lambda < 1, \end{cases}$$

where $\hat{W}^*(\lambda)$ is defined in (4.30) and $\hat{W}^+(\lambda)$ is defined as

$$\hat{W}^+(\lambda) = \begin{bmatrix} 0 & \gamma & 1-\gamma \\ \gamma & 1-\gamma & 0 \\ 1-\gamma & 0 & \gamma \end{bmatrix} \quad \text{with } \gamma = \frac{1}{2} + \sqrt{\frac{1}{12}(4\lambda^2 - 1)}. \tag{4.31}$$

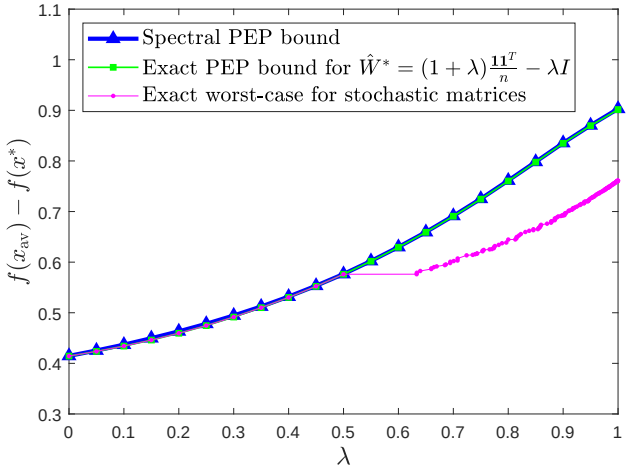


Fig. 4.3 Evolution with λ of the worst-case performance of $t = 10$ iterations of DGD with $n = 3$ agents and convex functions with bounded subgradients. The plot shows (i) the spectral worst-case performance (in blue), (ii) the exact worst-case performance for the symmetric generalized doubly stochastic matrix $\hat{W}^*$ from equation (4.30) (in green), and (iii) the exact worst-case performance for symmetric doubly stochastic matrices found based on an exhaustive exploration of such matrices used in the exact PEP formulation (in pink). This indicates the tightness of the spectral formulation of PEP for DGD with symmetric generalized doubly stochastic matrices, within numerical errors.

When $\lambda \geq 0.5$, $\hat{W}^+(\lambda)$ is a non-negative averaging matrix that always has eigenvalues $\{1, \lambda, -\lambda\}$ and leads to smaller worst-case errors than the general worst-case averaging matrix $\hat{W}^*(\lambda)$, with no restriction on its sign. For $\lambda \in [0.5, 0.62]$, matrices $\hat{W}^+(\lambda)$ is not the worst-case non-negative matrix, since $\hat{W}^*(0.5)$ leads to a larger error. This explains the plateau of the worst-case error that can be observed in Fig. 4.3 between these values of λ.

This shows that the spectral PEP bound is thus no longer tight for DGD with doubly stochastic matrices. However, as shown in Fig. 1.2 and further developed in Chapter 8, the resulting PEP bounds still outperform existing theoretical performance bounds from the literature. This observation has been confirmed for other values of t and n ($n = 3, 5, 7$, and $t = 10, 15$).

While the worst-case averaging matrix $\hat{W}^*(\lambda)$ (4.30) always has a complete topology, with possibly zero or negative elements on the diagonal, the worst-case non-negative averaging matrix can correspond to different topologies depending on λ. For example, for $n = 3$, the worst-case matrix $\hat{W}^+(\lambda)$ (4.31) has two non-zero edge weights (out of 3) for $\lambda > 0.62$.

4.6 Summary and conclusion

In summary, in this chapter, we have seen that the worst-case performance of t iterations of a decentralized optimization algorithm from $\mathcal{A}_D$ can be computed using an SDP reformulation of the Performance Estimation Problem (PEP) with a size growing with t and the number of agents n. When the averaging matrix W is given, the resulting PEP worst-case bound is tight but specific. We refer to such a PEP as the *exact PEP formulation*. When computing a worst-case valid for all averaging matrices from $\mathcal{W}_{(\lambda^-, \lambda^+)}$ (symmetric, generalized doubly stochastic, and with a range on the eigenvalues), the resulting PEP is a relaxation, which may provide non-tight performance bounds and is called the *spectral PEP formulation*. We have proved that some form of relaxation is inevitable if we want to obtain a convex formulation (see Theorem 4.2). The constraints we provide to represent the consensus steps in PEP allow for more general matrix steps that can mix components of vectors to which they apply (see Theorem 4.6). While this is not natural, it does not alter the convergence of pure consensus, and we do not expect it to be an important source of conservatism in performance limits. (see Proposition 4.11). Moreover, the tightness of the spectral PEP solutions can be verified a posteriori by recovering an instance

of worst-averaging matrix in $\mathcal{W}_{(\lambda^-,\lambda^+)}$. According to our experiments, the spectral PEP formulation for DGD appears to be tight for averaging matrices from $\mathcal{W}_{(\lambda^-,\lambda^+)}$ (which includes matrices with negative elements) but not when we restrict to (non-negative) doubly stochastic matrices. Finally, in our numerical experiment, we have also observed that the worst-case guarantees obtained with the spectral PEP formulation are independent of the number of agents n for various algorithms and settings. This observation motivates us to build other PEP formulations with a size independent of n, which are presented in Chapters 5 and 6.

To solve the puzzle
Gather similar pieces
Then take a step back.

5

Agent-independent relaxed PEP formulation

THE GOAL of this chapter is to develop a formulation of the Performance Estimation Problem (PEP) for decentralized optimization with a problem size independent of the number of agents in the network. The agent-dependent PEP formulation from Chapter 4 considers each agent individually, implying that its size grows with the number of agents n. However, we observed in our experiments (Section 4.5) that the results are independent of n for many classical algorithms and settings. This observation suggests that the same results could be obtained with PEPs that are independent of n, and motivates the work of this chapter, and of Chapter 6. The approach from this chapter leads to a relaxed PEP formulation, unlike the compact PEP formulation developed in Chapter 6, which will generally be preferred. Nevertheless, we present this relaxed PEP formulation because it allows us to observe that the separability of the global objective function F^{s_n} does not influence the worst-case performance of decentralized algorithms in many settings. This insight opens the way for further research in this direction.

The approach of this chapter is based on the fact that the size of an SDP PEP (3.6) depends on the number of variables considered and not on their dimension (see Chapter 3), so we can obtain an agent-independent representation by abstracting the variables and functions at the global level, see Section 5.1. This requires considering a generalized version of the decen-

tralized problem (2.4), where the separability of the global objective function F^{s_n} is neglected, which a priori corresponds to a relaxation, see Section 5.2. Then, as explained in Section 5.3, the resulting PEP problem admits a simple representation that only requires decomposing $\mathbf{x} \in \mathbb{R}^{nd}$ into two sets of components, respectively along the consensus subspace and along its orthogonal complement, as opposed to the n sets of components along each x_i used in Chapter 4 to exploit separability of the decentralized problem. While the size of the resulting PEP problem is independent of the number of agents n, the value of n can still appear in the problem as a parameter and impact the PEP guarantees. However, for many common algorithms and settings, the problem is totally independent of n.

In Section 5.4, we apply our new *agent-independent relaxed PEP formulation* to different decentralized methods such as DGD and DIGing and obtain performance bounds that are valid for any network size n. Moreover, the bounds are tight since they match those computed with the spectral agent-dependent PEP formulation from Chapter 4. This means that the relaxation of the separability of the global objective function does not impact the worst-case performance in that cases, and thus the worst-case performance of these algorithms only depends on the global function properties.

This chapter is based on the paper [CH22].

5.1 PEP with global vector and function representation

The SDP PEP reformulation (3.6) has the Gram matrix of scalar products (3.5) as variable. Hence, the SDP problem size depends on the number of vectors involved in the problem, but not on their dimension. Therefore, in decentralized optimization, an intuitive idea to obtain a PEP formulation with size independent of the number of agents n is to consider fewer vectors, but lying in a larger dimensional space, by stacking the local vectors, e.g., $x_1^k, \dots, x_n^k \in \mathbb{R}^d$, in one vector, $\mathbf{x} \in \mathbb{R}^{nd}$, see Section 2.7 for details on the notation. The other variable of the SDP PEP is the function values vectors $\mathbf{f}$ (3.4), whose size grows with n in the agent-dependent formulation of Chapter 4 because it considers each local function individually (4.3). To have a problem size independent of the number of agents n, we thus need to only consider the global separable function F^{s_n} from the decentralized problem (2.4). Using this global representation, we can write the following high-level PEP description,

$$
\begin{aligned}
w(\mathcal{S}_n) = \max_{F^{s_n}, \mathbf{x}^0, \ldots, \mathbf{x}^t, \mathbf{x}^*} \quad & \mathcal{P}\left(F^{s_n}, \mathbf{x}^0, \ldots, \mathbf{x}^t, \mathbf{x}^*\right) \\
\text{s.t.} \quad & F^{s_n} \in \mathcal{F}^{s_n}, \\
& \mathbf{x}^1, \ldots, \mathbf{x}^k \text{ vectors of iterates from method } A \in \mathcal{A}_D \text{ applied on } F^{s_n}, \\
& \mathbf{x}^* \text{ optimal solution of (2.4)}, \\
& \mathbf{x}^0 \text{ satisfies } \mathcal{I},
\end{aligned}
\tag{5.1}
$$

where we define $\mathcal{F}^{s_n}$ as the class of functions that are separable into n functions from a given class $\mathcal{F}$.

Definition 5.1 ($\mathcal{F}^{s_n}$)**.** Let $\mathcal{F}(\mathbb{R}^d)$ be a class of functions from $\mathbb{R}^d$ to $\mathbb{R}$, e.g., $\mathcal{F}_R$ or $\mathcal{F}_{\mu,L}$ from Definitions 3.2 and 3.3. A function $F : \mathbb{R}^{nd} \to \mathbb{R}$ is in $\mathcal{F}^{s_n}(\mathbb{R}^{nd})$ if there are functions $f_i \in \mathcal{F}(\mathbb{R}^d)$, for all $i = 1, \ldots, n$, such that F is separable as

$$
F(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} f_i(x_i),
$$

where $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^{nd}$.

This PEP with global representation for decentralized optimization only applies to settings that are not agent-specific. For example, we cannot impose initial conditions on a single agent, or a subset of agents, and we cannot analyze the performance of the worst agent, or the one of a specific subset of agents. Indeed, such settings cannot be described with the global function F^{s_n} or the global vectors, e.g., $\mathbf{x}$.

To obtain an SDP PEP formulation, we need to formulate the performance measure $\mathcal{P}$ and the constraints of (5.1) in a Gram-representable manner. However, we have no idea how to represent the class of separable functions $\mathcal{F}^{s_n}$, without distinguishing the different local functions. Therefore, we relax the separability assumption of the objective function $F^{s_n} : \mathbb{R}^{nd} \to \mathbb{R}$, by considering a general real function $F : \mathbb{R}^{nd} \to \mathbb{R}$ of the same class.

5.2 Relax the separability of the objective function

When relaxing the separability of the objective function, we generalize the decentralized problem (2.4). The generalized problem can be written in the following manner, using Definition 2.6 of the consensus subspace, as well

as the vector notations defined in Section 2.7,

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^{nd}} \quad & F(\mathbf{x}), \\ \text{such that } & \mathbf{x} \in \mathcal{C}, \end{aligned} \tag{5.2}$$

where $F : \mathbb{R}^{nd} \to \mathbb{R}$ is a real function from a given class of functions $\mathcal{F}$. We develop results for classes of functions $\mathcal{F}_{\mu,L}$ (Definition 3.2) and $\mathcal{F}_R$ (Definition 3.3), and their corresponding classes of separable functions $\mathcal{F}_{\mu,L}^{s_n}$ and $\mathcal{F}_R^{s_n}$ (Definition 5.1). The results can be extended to the other classes of functions that are Gram-representable.

We now argue that the worst-case performance of a decentralized algorithm solving problem (2.4) can be bounded by its performance on problem (5.2). The argument relies on the following proposition which follows from Definition 5.1 combined with Definitions 3.2 and 3.3.

Proposition 5.2 (Relaxation of separability)**.** *The sets of separable functions (Definition 5.1) are such that*

$$\mathcal{F}_R^{s_n} \subseteq \mathcal{F}_{\frac{R}{\sqrt{n}}}, \text{ and } \mathcal{F}_{\mu,L}^{s_n} \subseteq \mathcal{F}_{\frac{\mu}{n},\frac{L}{n}}.$$

Proof. From Definition 5.1, we know that $F \in \mathcal{F}_R^{s_n}$ is separable into

$$F(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(x_i) \qquad \text{with } f_i \in \mathcal{F}_R.$$

By Definition 3.3 of $\mathcal{F}_R$, each function $f_i \in \mathcal{F}_R$ is convex and with bounded subgradients $\|g_i\|^2 \leq R^2$. Therefore, F is convex because it is the sum of convex functions, and its subgradients $\mathbf{g}$ are bounded by

$$\|\mathbf{g}\|^2 = \frac{1}{n^2}\sum_{i}^{n} \|g_i\|^2 \leq \frac{R^2}{n},$$

meaning that $F \in \mathcal{F}_{\frac{R}{\sqrt{n}}}$. Similarly, for $F \in \mathcal{F}_{\mu,L}^{s_n}$, since it is separable into

$$F(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(x_i) \qquad \text{with } f_i \in \mathcal{F}_{\mu,L},$$

and that each function $f_i \in \mathcal{F}_{\mu,L}$ satisfies constraints (2.2) and (2.3), we have that, for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{nd}$,

$$\text{and } \frac{1}{n}\sum_{i=1}^n f_i(y_i) \leq \frac{1}{n}\sum_{i=1}^n f_i(x_i) + \frac{1}{n}\sum_{i=1}^n \nabla f_i^T(x_i)(y_i - x_i) + \frac{L}{2n}\sum_{i=1}^n \|y_i - x_i\|^2,$$

$$\Leftrightarrow \quad F(\mathbf{y}) \leq F(\mathbf{x}) + \nabla F^T(\mathbf{x})(\mathbf{y} - \mathbf{x}) + \frac{L}{2n}\|\mathbf{y} - \mathbf{x}\|^2$$

$$\frac{1}{n}\sum_{i=1}^n f_i(y_i) \geq \frac{1}{n}\sum_{i=1}^n f_i(x_i) + \frac{1}{n}\sum_{i=1}^n \nabla f_i^T(x_i)(y_i - x_i) + \frac{\mu}{2n}\sum_{i=1}^n \|y_i - x_i\|^2,$$

$$\Leftrightarrow \quad F(\mathbf{y}) \geq F(\mathbf{x}) + \nabla F^T(\mathbf{x})(\mathbf{y} - \mathbf{x}) + \frac{\mu}{2n}\|\mathbf{y} - \mathbf{x}\|^2$$

meaning that $F \in \mathcal{F}_{\frac{\mu}{n},\frac{L}{n}}$. □

Proposition 5.2 means that any optimization problem, such as a PEP, with constraint $F \in \mathcal{F}_{\frac{R}{\sqrt{n}}}$ (resp. $F \in \mathcal{F}_{\frac{\mu}{n},\frac{L}{n}}$) is a relaxation of the one with constraint $F \in \mathcal{F}_R^{\mathcal{S}_n}$ (resp. $F \in \mathcal{F}_{\mu,L}^{\mathcal{S}_n}$). We can thus compute upper bounds on the worst-case performance of decentralized algorithms for solving (2.4) by analyzing their performance on the general problem (5.2). This consists, a priori, in a relaxation since we no longer exploit the separability of the objective function, but we observe the tightness of the obtained bounds in the numerical results of Section 5.4

To analyze the performance of an algorithm on problem (5.2) using an SDP PEP, we need, in particular, to have Gram-representable constraints to characterize its optimal solutions. Therefore, we need to express the consensus constraint $\mathbf{x} \in \mathcal{C}$ in a Gram-representable form. To do this, we consider the problem in another coordinate system, which relies on the decomposition of $\mathbb{R}^{nd}$ into subspaces $\mathcal{C}$ and $\mathcal{C}^\perp$.

5.3 Change of variables and new coordinates system

By definition, we have that $\mathbb{R}^{nd} = \mathcal{C} \oplus \mathcal{C}^\perp$, so that any vector $\mathbf{x} \in \mathbb{R}^{nd}$ can be uniquely decomposed as

$$\mathbf{x} = \mathbf{x}_{/\!/} + \mathbf{x}_\perp, \qquad \text{with } \mathbf{x}_{/\!/} \in \mathcal{C} \text{ and } \mathbf{x}_\perp \in \mathcal{C}^\perp.$$

Therefore, a vector $\mathbf{x} \in \mathcal{C}$ if and only if $\mathbf{x}_\perp = 0$. Moreover, as we will see, a consensus step with a symmetric and generalized doubly stochastic averaging matrix does not impact $\mathbf{x}_{/\!/}$, but only contracts $\mathbf{x}_\perp$. These observations motivate the following change of variable.

Definition 5.3 (Change of variable ${}_b\mathbf{x}$)**.** Let $Q_{/\!/} = \frac{1}{\sqrt{n}}(\mathbf{1}_n \otimes I_d)$ be the orthonormal basis of $\mathcal{C}$ and $Q_\perp \in \mathbb{R}^{nd\times(n-1)d}$ the orthonormal basis of $\mathcal{C}^\perp$. For any $\mathbf{x} \in \mathbb{R}^{nd}$, we define ${}_b\mathbf{x} \in \mathbb{R}^{nd}$ as

$$ {}_b\mathbf{x} = \begin{bmatrix} \overline{x} \\ x_\perp \end{bmatrix} = \frac{1}{\sqrt{n}} \begin{bmatrix} Q_{/\!/}^T \\ Q_\perp^T \end{bmatrix} \mathbf{x} = \frac{1}{\sqrt{n}} Q^T \mathbf{x}, \tag{5.3} $$

where $\overline{x} = \frac{1}{n}\sum_{i=1}^n x_i \in \mathbb{R}^d$ and $x_\perp \in \mathbb{R}^{(n-1)d}$. By definition, the matrix $Q = [Q_{/\!/} \;\; Q_\perp]$ is orthogonal, i.e., $Q^TQ = QQ^T = I$, and the change of variable can thus be inverted *via*

$$ \mathbf{x} = \sqrt{n} Q \, {}_b\mathbf{x}. \tag{5.4} $$

This change of variable can be interpreted as a change of coordinate system: $\mathbf{x}$ describes the coordinates in the standard basis and ${}_b\mathbf{x}$ its coordinates in a new basis b, with $\overline{x}$ the part of the coordinates describing the position along the consensus subspace $\mathcal{C}$, and $x_\perp$ the part describing the position along the orthogonal complement $\mathcal{C}^\perp$. We use factor $\frac{1}{\sqrt{n}}$ in the change of variable (5.3) to ensure that $\overline{x}$ is indeed the agent average. Using Definition 5.3, we can easily express the constraint $\mathbf{x} \in \mathcal{C}$.

Proposition 5.4. $\mathbf{x} \in \mathcal{C}$ *iff* $x_\perp = 0$.

Proof. We obtain this from (5.4),

$$ \mathbf{x} = \sqrt{n} Q_{/\!/} \overline{x} + \sqrt{n} Q_\perp x_\perp, $$

where the first term is in $\mathcal{C}$ and the second is in $\mathcal{C}^\perp$. □

We now review the impacts of this change of variable on the generalized problem, the formulation of a decentralized algorithm, and the performance estimation problem.

5.3.1 Changes in the Generalized Decentralized Problem

We define a new function $\tilde{F}({}_b\mathbf{x}) : \mathbb{R}^{nd} \to \mathbb{R}$ depending on the new variable ${}_b\mathbf{x}$ as

$$ \tilde{F}({}_b\mathbf{x}) = F(\sqrt{n} Q \, {}_b\mathbf{x}) = F(\mathbf{x}). \tag{5.5} $$

Using our change of variable and function, as well as Proposition 5.4, problem (5.2) becomes

$$\begin{aligned} \min_{{}_b\mathbf{x} \in \mathbb{R}^{nd}} \quad & \tilde{F}({}_b\mathbf{x}), \\ \text{such that } & x_\perp = 0. \end{aligned} \tag{5.6}$$

Let us analyze what are the properties of the function $\tilde{F}$ and its gradient, in order to formulate the performance estimation problem for algorithms solving this problem.

Proposition 5.5 (Gradients of $\tilde{F}$)**.**
Let $\nabla\tilde{F}({}_b\mathbf{x})$ be the gradient of $\tilde{F}$ at point ${}_b\mathbf{x}$ and $\nabla F(\mathbf{x})$ the gradient of F at point $\mathbf{x}$. There holds

$$\nabla\tilde{F}({}_b\mathbf{x}) = \sqrt{n}Q^T\nabla F(\mathbf{x}). \tag{5.7}$$

Proof. Using the chain rule, direct computation shows

$$\nabla_{{}_b\mathbf{x}}\tilde{F}({}_b\mathbf{x}) = \nabla_{{}_b\mathbf{x}}F(\sqrt{n}Q\,{}_b\mathbf{x}) = \sqrt{n}Q^T\nabla_{\mathbf{x}}F(\mathbf{x}),$$

where indices for ∇ indicate which variable is considered in the derivatives. □

The gradient $\nabla\tilde{F}({}_b\mathbf{x})$ can be decomposed into two parts that respectively describe the variation of the value of $\tilde{F}({}_b\mathbf{x})$ with the consensus and orthogonal components:

$$\nabla\tilde{F}({}_b\mathbf{x}) = \begin{bmatrix} \nabla_{/\!/}\,\tilde{F}({}_b\mathbf{x}) \\ \nabla_\perp\tilde{F}({}_b\mathbf{x}) \end{bmatrix}. \tag{5.8}$$

Proposition 5.5 implies the following corollary.

Corollary 5.6. *The norm of the gradient of $\tilde{F}$ at ${}_b\mathbf{x}$ can be computed as*

$$\|\nabla\tilde{F}({}_b\mathbf{x})\|_2 = \sqrt{n}\|\nabla F(\mathbf{x})\|_2.$$

Moreover, if $F \in \mathcal{F}^{s_n}$, so that it is separable as $F(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^n f_i(x_i)$, then $\nabla\tilde{F}({}_b\mathbf{x})$ is the coordinate vector in basis b of the vector $\mathbf{g}^T = \left[\nabla f_1^T(x_1)\dots\nabla f_n^T(x_n)\right]$, i.e.,

$$\nabla\tilde{F}({}_b\mathbf{x}) = \frac{1}{\sqrt{n}}Q^T\mathbf{g} = {}_b\mathbf{g}.$$

In particular, we have that $\nabla_{/\!/}\,\tilde{F}({}_b\mathbf{x}) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(x_i)$.

Proposition 5.7 (Optimal Conditions of (5.6))**.** *The vector ${}_b\mathbf{x}^*$ is an optimal solution for problem (5.6) if and only if*

$$x_\perp^* = 0 \quad \textit{and} \quad \nabla_{/\!/}\tilde{F}({}_b\mathbf{x}^*) = 0.$$

Proof. By definition of problem (5.6), $x_\perp = 0$ for any feasible solution and thus the problem is equivalent to

$$\min_{{}_b\mathbf{x} \in \mathbb{R}^{nd}} \quad \tilde{F}\left(\begin{bmatrix}\overline{x}\\ 0\end{bmatrix}\right),$$

from which the result follows by convexity of $\tilde{F}$. □

Proposition 5.8 (Properties of $\tilde{F}$)**.** *Let $\mathcal{F}_R$ be the class of Lipschitz convex functions (Definition 3.3) and $\mathcal{F}_{\mu,L}$ the class of L-smooth and μ-strongly convex functions (Definition 3.2). In general,*

(i) If $F \in \mathcal{F}_{\frac{R}{\sqrt{n}}}$, then $\tilde{F} \in \mathcal{F}_R$;

(ii) If $F \in \mathcal{F}_{\frac{\mu}{n},\frac{L}{n}}$, then $\tilde{F} \in \mathcal{F}_{\mu,L}$.

In cases where F is separable, this becomes

(iii) If $F \in \mathcal{F}_R^{Sn}$, then $\tilde{F} \in \mathcal{F}_R$;

(iv) If $F \in \mathcal{F}_{\mu,L}^{Sn}$, then $\tilde{F} \in \mathcal{F}_{\mu,L}$.

Proof. (i) If $F \in \mathcal{F}_{\frac{R}{\sqrt{n}}}$, then $\|\nabla F(\mathbf{x})\|_2 \leq \frac{R}{\sqrt{n}}$ for all $\mathbf{x}$ and using Corollary 5.6, we have $\|\nabla\tilde{F}({}_b\mathbf{x})\|_2 \leq R$. (ii) Starting from Definition 3.2 for $F \in \mathcal{F}_{\frac{\mu}{n},\frac{L}{n}}$, we can use equation (5.4), (5.5) and (5.7) to express it in term of variables ${}_b\mathbf{x}$ and ${}_b\mathbf{y}$, the function $\tilde{F}$ and its gradients. Then, using orthogonality of Q, we recover conditions for $\tilde{F} \in \mathcal{F}_{\mu,L}$.
Finally, (iii) and (iv) directly follow from (i) and (ii) combined with Proposition 5.2. □

5.3.2 Changes in the Decentralized Algorithms

After applying the changes of variables (5.4), an algorithm for solving problem (5.2) will be suited for solving problem (5.6). The impact of these changes of variables can be treated independently for each of the three possible operations of a decentralized algorithm from $\mathcal{A}_D$.

Gradient: Using Corollary 5.6, the computation of the vector $\mathbf{g}$ of local gradients of function F at point $\mathbf{x}$ simply becomes the computation of $\nabla \tilde{F}({}_b\mathbf{x}) = {}_b\mathbf{g}$.

Linear combinations: Since Q is orthogonal, the changes of variables do not modify the linear combination operations, e.g., applying (5.4) to the combination $\mathbf{z} = \alpha\mathbf{x} + \beta\mathbf{y}$ gives

$$\sqrt{n}Q\,{}_b\mathbf{z} = \sqrt{n}Q(\alpha\,{}_b\mathbf{x} + \beta\,{}_b\mathbf{y}) \;\Leftrightarrow\; {}_b\mathbf{z} = \alpha\,{}_b\mathbf{x} + \beta\,{}_b\mathbf{y}.$$

Consensus: Theorem 5.9 analyzes the effects of a consensus step on the new variables.

Theorem 5.9 (Consensus step in basis b)**.** *Let $W \in \mathbb{R}^{n\times n}$ be a symmetric generalized doubly stochastic matrix. The consensus step $\mathbf{y} = (W \otimes I_d)\mathbf{x}$, is expressed in basis b as*

$${}_b\mathbf{y} = \begin{bmatrix} \overline{y} \\ y_\perp \end{bmatrix} = \begin{bmatrix} \overline{x} \\ \tilde{W}x_\perp \end{bmatrix}, \tag{5.9}$$

where $\tilde{W} \in \mathbb{R}^{(n-1)d\times(n-1)d}$ is a symmetric matrix with the same eigenvalues as W with a multiplicity d times larger, except for $\lambda_1 = 1$ that is not an eigenvalue of $\tilde{W}$.

Proof. When applying changes of variables (5.4) in the standard consensus step (2.19), since Q is orthogonal, we have

$${}_b\mathbf{y} = {}_bW\,{}_b\mathbf{x} \quad \text{with} \quad {}_bW = Q^T(W \otimes I_d)Q, \tag{5.10}$$

where ${}_bW \in \mathbb{R}^{nd\times nd}$ is the averaging matrix for the consensus expressed in the basis b and has the same eigenvalues as $(W \otimes I_d)$ by similarity. By definition, matrix $(W \otimes I_d)$ is symmetric, generalized doubly stochastic, has the same eigenvalues as W with a multiplicity d times larger for each of them, and it can thus be written using the eigen-decomposition:

$$W \otimes I_d = V\Lambda V^T = \begin{bmatrix} V_{/\!/} & V_\perp \end{bmatrix} \begin{bmatrix} I_d & 0 \\ 0 & \Lambda_\perp \end{bmatrix} \begin{bmatrix} V_{/\!/}^T \\ V_\perp^T \end{bmatrix},$$

where $V_{/\!/} = Q_{/\!/} = \frac{1}{\sqrt{n}}(\mathbf{1}_n \times I_d)$ are the orthonormal eigenvectors corresponding to the d eigenvalues equal to 1 and $V_\perp$ are the orthonormal eigenvectors corresponding to other eigenvalues $\Lambda_\perp$. All the eigenvectors are orthogonal to each other and therefore, since $V_{/\!/} = Q_{/\!/}$, the matrix ${}_bW$ from (5.10) is given by

$$ {}_bW = \begin{bmatrix} Q_{//}^T \\ Q_\perp^T \end{bmatrix} \begin{bmatrix} V_{//} & V_\perp \end{bmatrix} \begin{bmatrix} I_d & 0 \\ 0 & \Lambda_\perp \end{bmatrix} \begin{bmatrix} V_{//}^T \\ V_\perp^T \end{bmatrix} \begin{bmatrix} Q_{//} & Q_\perp \end{bmatrix} = \begin{bmatrix} I_d & 0 \\ 0 & \tilde{W} \end{bmatrix}, \quad (5.11) $$

where $\tilde{W} = Q_\perp^T V_\perp \Lambda_\perp V_\perp^T Q_\perp$ and is thus symmetric. Eigenvalues of $\tilde{W}$ are the ones in $\Lambda_\perp$ since ${}_bW$ and $(W \otimes I_d)$ have the same eigenvalues. Finally, we find (5.9) using expression (5.11) for ${}_bW$ in the consensus step (5.10). □

Theorem 5.9 formalizes the separate effects of a consensus step that we discussed at the beginning of Section 5.3. We now have access to the problem, its properties, and any decentralized algorithm $A \in \mathcal{A}_D$ written using the new variables. There only remains to formulate the PEP that evaluates the worst-case performance of A for solving problem (5.6) with a given class of functions.

5.3.3 Changes in the Performance Estimation Problem

This subsection reviews the effect of the changes of variables (5.4) in a PEP (3.2) applied to a decentralized algorithm for solving problem (5.6). We explain how such PEPs can be formulated as SDPs that are independent of the dimension of vector ${}_b\mathbf{x} \in \mathbb{R}^{nd}$, and thus independent of n.

First, the optimal conditions for (5.6) are given in Proposition 5.7. We observe that they require the use of specific components $x_\perp$ and $\nabla_{//}\tilde{F}$, independently of the full vectors ${}_b\mathbf{x}$ and $\nabla\tilde{F}$. This separation of consensus coordinates and orthogonal coordinates also occurs in the effect of consensus steps in Theorem 5.9. To be able to use such components independently of each other, we split the vectors ${}_b\mathbf{x}^k$ and ${}_b\mathbf{g}^k = \nabla\tilde{F}({}_b\mathbf{x}^k)$ in two parts in the SDP PEP formulation, described in Section 3.3, and we consider the scalar product of each part separately. This results then in two Gram matrices:

$$ G_{//} = P_{//}^T P_{//} \quad \text{and} \quad G_\perp = P_\perp^T P_\perp, \quad (5.12) $$

where $P_{//} \in \mathbb{R}^{d \times 2K+2}$ and $P_\perp \in \mathbb{R}^{(n-1)d \times 2K+2}$ contains respectively the consensus and the orthogonal components of vectors ${}_b\mathbf{x}^k$ and ${}_b\mathbf{g}^k$. By definition, $G_{//} \succeq 0$ and $G_\perp \succeq 0$ and the Gram matrices have identical sizes, independent of the number of agents n. They can possibly be combined into a single matrix $G \succeq 0$ as

$$ G = \begin{bmatrix} G_{//} & 0 \\ 0 & G_\perp \end{bmatrix}. $$

This modification still allows writing all the constraints involving scalar products between the iterates ${}_b\mathbf{x}$ or the gradient ${}_b\mathbf{g}$, e.g., the usual interpo-

lations conditions for imposing that $\tilde{F} \in \mathcal{F}$, using the fact that

$$({}_b\mathbf{x})^T({}_b\mathbf{x}) = \overline{x}^T\overline{x} + x_\perp^T x_\perp, \qquad ({}_b\mathbf{g})^T({}_b\mathbf{g}) = g_{//}^T g_{//} + g_\perp^T g_\perp,$$
$$\text{and} \quad ({}_b\mathbf{x})^T({}_b\mathbf{g}) = \overline{x}^T g_{//} + x_\perp^T g_\perp.$$

The function values vector (3.4), used as the second variable in SDP PEP, contains function values of $\tilde{F}$:

$$\tilde{\mathbf{f}} = [\tilde{F}({}_b\mathbf{x}^0), \dots \tilde{F}({}_b\mathbf{x}^t), \tilde{F}({}_b\mathbf{x}^*)]. \tag{5.13}$$

We can therefore reformulate the PEP problem (5.1) into an SDP PEP by relaxing the separability assumption and applying a change of variables. The variables of the resulting SDP PEP are $G_{//} \succeq 0$, $G_\perp \succeq 0$ (5.12) and $\tilde{\mathbf{f}}$ (5.13). All the elements of the problem (objective performance measure and constraints) that are not agent-specific can be written linearly in terms of these variables, i.e., they are linearly Gram-representable, can thus be embedded in the SDP PEP formulation, see Proposition 3.7.

Indeed, the usual interpolation constraints, e.g., those for $\mathcal{F}_{\mu,L}$ (Proposition 3.4) or $\mathcal{F}_R$ (Proposition 3.5), are well linearly Gram-representable because they are expressed as linear combinations of scalar products between $\overline{x}^k, g_{//}^k$ and between $x_\perp^k, g_\perp^k$.

Concerning the algorithmic steps constraints, we need to decouple them into constraints for the consensus components and for the orthogonal components. Let us analyze separately each possible operation of a decentralized algorithm.

Gradient and linear combinations: The gradient sampling of $\tilde{F}$ and the linear combinations of vectors in basis b can directly be decoupled for the consensus and the orthogonal components, see (5.8) and (5.3). These constraints are linear equalities and are thus linearly Gram-representable (see Section 4.2 for details).

Consensus: As in Chapter 4, to obtain general worst-case guarantees, valid for an entire class of averaging matrices, we formulate PEPs with consensus steps of the form

$$\mathbf{y} = (W \otimes I_d)\mathbf{x}, \qquad \text{with } W \in \mathcal{W}_{(\lambda^-,\lambda^+)},$$

where $\mathcal{W}_{(\lambda^-,\lambda^+)}$ (4.10) is the set of symmetric and generalized doubly stochastic averaging matrices with a given range of eigenvalues, excluding $\lambda_1 = 1$. The corresponding separate updates for the consensus and orthogonal parts of the vectors are given in Theorem 5.9. For the consensus components, it corresponds to a linear equality, and so it is linearly Gram-

representable. For the orthogonal components, it corresponds to a linear update with an unknown symmetric matrix $\tilde{W} \in \mathbb{R}^{(n-1)d\times(n-1)d}$ with the given of range eigenvalues:

$$y_\perp^k = \tilde{W} x_\perp^k \qquad \lambda^- \succeq \tilde{W} \succeq \lambda^+, \tag{5.14}$$

where the notations $A \succeq B$ denote positive semidefiniteness of $A - B$. We stress that the range $[\lambda^-, \lambda^+]$ of eigenvalues is known in our performance analysis, but does not necessarily need to be known for the execution of the algorithms. The representation of such a symmetric linear operator in PEP has been studied in [BHG23], which provides Gram-representable interpolation constraints for it. The set of update constraints (5.14) for $k = 0, \dots, t$, is replaced in PEP by the set of constraints on pairs $\{(y_\perp^k, x_\perp^k)\}$ provided in Theorem 5.10 below. These constraints allow the PEP to find worst-case valid over an entire spectral class of averaging matrix.

Theorem 5.10 (Symmetric linear operator interpolability [BHG23])**.** *Let $X \in \mathbb{R}^{q\times t}$ and $Y \in \mathbb{R}^{q\times t}$, there is a symmetric matrix $\tilde{W} \in \mathbb{R}^{q\times q}$ such that*

$$Y = \tilde{W} X \quad \text{and} \quad \lambda^- \succeq \tilde{W} \succeq \lambda^+,$$

if, and only if,

$$X^T Y = Y^T X,$$
$$(Y - \lambda^- X)^T (Y - \lambda^+ X) \preceq 0.$$

The proof of Theorem 5.10 can be found in [BHG23, Theorem 3.3].

The last important ingredients of PEP are the initial conditions and the performance criterion. Using $(\overline{x}^k, x_\perp^k, g_{/\!/}^k, g_\perp^k, \tilde{F}^k)_{k=0,\dots,t,*}$, as well as all their properties studied in previous subsections, we can formulate any metric that is not agent-specific. Propositions 5.11 and 5.12 give two examples of initial conditions. Metrics used for initial conditions can also be used as performance criteria after t iterations.

Proposition 5.11. *The initialization of variables $x_i^0 \in \mathbb{R}^d$, for $i = 1, \dots, n$ such that*

$$x_i^0 = x^0 \text{ for all } i, \quad \text{and} \quad \|x^0 - x^*\|_2^2 \leq D^2,$$

where $x^* \in \mathbb{R}^d$ is an optimal solution of (2.1), is equivalent to the following linearly Gram-representable constraints

$$x_\perp^0 = 0, \quad \text{and} \quad \|\overline{x}^0 - \overline{x}^*\|_2^2 \leq D^2,$$

where ${}_b\mathbf{x}^* = \begin{bmatrix} \overline{x}^* \\ 0 \end{bmatrix} \in \mathbb{R}^{nd}$ is an optimal solution of (5.6).

Proof. All the agents start with the same x_i^0, which means that $\mathbf{x}^0 \in \mathcal{C}$ and thus $x_\perp^0 = 0$, see Proposition 5.4. We also have, by definition, that $\overline{x}^0 = x^0$ and $\overline{x}^* = x^*$. These conditions are linearly Gram-representable since they involve linear combinations of scalar products of vectors $x_\perp^0$, $\overline{x}^0$, and $\overline{x}^*$. □

Proposition 5.12. *The initialization of variables $x_i^0 \in \mathbb{R}^d$, for $i = 1, \dots, n$ such that*

$$\frac{1}{n}\sum_{i=1}^{n} \|x_i^0 - x^*\|_2^2 \leq D^2, \tag{5.15}$$

where $x^* \in \mathbb{R}^d$ is an optimal solution of (2.1), is equivalent to the following linearly Gram-representable constraints

$$\|{}_b\mathbf{x}^0 - {}_b\mathbf{x}^*\|_2^2 = \|\overline{x}^0 - \overline{x}^*\|_2^2 + \|x_\perp^0 - x_\perp^*\|_2^2 \leq D^2, \tag{5.16}$$

where ${}_b\mathbf{x}^* \in \mathbb{R}^{nd}$ is an optimal solution of (5.6).

Proof. Using change of variables (5.4), the initial condition (5.15) becomes

$$\frac{1}{n}\sum_{i=1}^{n} \|x_i^0 - x^*\|_2^2 = \frac{1}{n}\|\mathbf{x}^0 - \mathbf{x}^*\|_2^2 = \|{}_b\mathbf{x}^0 - {}_b\mathbf{x}^*\|_2^2 \leq D^2.$$

We then obtain (5.16) by definition of the Euclidean norm. This condition is linearly Gram-representable since it involves a linear combination of scalar products of vectors $x_\perp^0$, $\overline{x}^0$, $x_\perp^*$ and $\overline{x}^*$. □

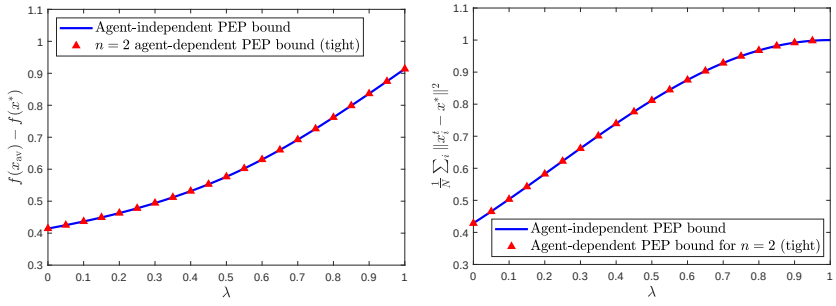
This last example can also apply for metric $\sum_{i=1}^n \|x_i^0 - x^*\|_2^2$, but in that case, n would appear as a scaling parameter for the formulation in basis b.

We now have all the ingredients to build an SDP PEP formulation with $G_{/\!/}$, $G_\perp$ (5.12) and $\tilde{\mathbf{f}}$ (5.13) as variables, with a size independent of the number of agents n and that compute a bound on the worst-case performance of any decentralized algorithm $A \in \mathcal{A}_D$ to solve problem (2.4) for different classes of functions (e.g., $\mathcal{F}_R$ and $\mathcal{F}_{\mu,L}$). The provided bound is valid for any averaging matrix that is symmetric, generalized doubly stochastic, and with a given range of eigenvalues, i.e., any matrix from $\mathcal{W}_{(\lambda^-,\lambda^+)}$ (4.10). The bound may scale with n as a parameter, but we can often choose the initial conditions and the performance criterion to obtain a bound totally independent of n, see Chapter 6 for details on the specific conditions under which this is possible.

5.4 Numerical comparison with the agent-dependent PEP

Implementation We implement this agent-independent relaxed PEP formulation for analyzing the performance of t iterations of DGD and DIGing with a constant step-size and averaging matrices from the spectral class $\mathcal{W}_{(\lambda^-,\lambda^+)}$. The implementation is in Matlab and relies on the YALMIP toolbox and the Mosek solver. We refer to Chapter 7 for more details on the implementation and practical use of this PEP formulation, and to the GitHub repository Performance-Estimation-Problems-for-distributed-optimization for all the code examples with practical implementation of our PEP formulations for distributed optimization.

Comparison We consider a class of averaging matrices $\mathcal{W}_{(-\lambda,\lambda)}$ (4.10), with $\lambda \in [0,1)$, that contains symmetric generalized doubly stochastic matrices, with a range of eigenvalues $[-\lambda,\lambda]$ that holds for all eigenvalues, except for $\lambda_1 = 1$. In Fig. 5.1, we compare the result of the spectral agent-dependent PEP formulation for $n = 2$ agents, from Chapter 4, and our new agent-independent relaxed PEP formulation. The figure shows the evolu-



(a) DGD, $f_i \in \mathcal{F}_R$ **(b)** DIGing, $f_i \in \mathcal{F}_{\mu,L}$

Fig. 5.1 Evolution with λ of worst-case performance bounds of $t = 10$ iterations of DGD and DIGing with averaging matrices from $\mathcal{W}_{(-\lambda,\lambda)}$ (4.10). Each plot shows two bounds that coincide: an agent-independent PEP bound obtained with our new relaxed PEP formulation (in blue) and an agent-dependent PEP bound for $n = 2$ from Chapter 4 (in red). Here are the specific settings of each plot: (a) DGD with step-size $\alpha = \frac{1}{\sqrt{t}}$, initial conditions from Proposition 5.11 with $D = 1$ and $f_i \in \mathcal{F}_R$ with $R = 1$, for all $i \in \mathcal{V}$; (b) DIGing with $\alpha = 0.44(1-\lambda)^2$, initial conditions from Proposition 5.12 with $D = 1$ and $f_i \in \mathcal{F}_{\mu,L}$ with $\mu = 0.1$, $L = 1$, for all $i \in \mathcal{V}$.

tion of the performance bounds of $t = 10$ iterations of DGD and DIGing with the value of λ, characterizing the range of eigenvalues. For comparison purposes, we chose to use the same settings as in Section 4.5. While the value of n can still appear as a parameter in the new relaxed PEP formulation, it totally disappears from the problem in the settings used here. Therefore, this relaxed PEP formulation leads to performance bounds that are valid for any value of $n \geq 2$ by construction, and not only by a posteriori observations, as is the case for the agent-dependent PEP formulation from Chapter 4.

We observe that both for DGD and DIGing, the bound obtained with our new agent-independent relaxed PEP formulation matches the one obtained with the agent-dependent PEP formulation with $n = 2$ agents from Chapter 4. This matching also holds for other values of $n \geq 2$ since the result of the agent-dependent PEP is constant with n in these settings (see Fig. 4.2). This means that

(i) The agent-independent relaxed PEP formulation is numerically tight for DGD and DIGing with averaging matrices from $\mathcal{W}_{(-\lambda,\lambda)}$ since the tightness of the agent-dependent formulation has been verified a posteriori in these cases (see Section 4.5).

(ii) The relaxation of the separability of the objective function, which we have introduced in Section 5.2, has no impact on the PEP results for DGD and DIGing in such settings.

The fact that the separability of the objective function does not seem to play a role in the worst-case performance analysis of a decentralized algorithm, such as DGD or DIGing, is quite interesting and would merit a deeper analysis. In particular, it means that the worst-case performance of the algorithms is not directly impacted by the properties of the local functions, but only by those of the global function. This feature is particularly desirable, as it enables the algorithm to guarantee more general worst-case performance, that holds, for example, even when some local functions are not well conditioned, or even non-smooth or non-convex, but the global function still has good properties and conditioning (e.g., L-smooth and μ strongly convex).

As future work, it would be interesting to prove the settings for which the equivalence between the agent-independent relaxed PEP formulation developed in this chapter and the spectral agent-dependent formulation from Chapter 4 holds. Indeed, this would provide settings under which we know that the performance can be analyzed on general non-separable

functions, without impact on the worst-case guarantees. Therefore, this could also help us understand what makes an algorithm robust to heterogeneity in local function types or conditioning. One direction would be to check if a separable function $F^{s_n} \in \mathcal{F}^{s_n}$ can be built from any solution $G_{/\!/}$, $G_{\perp}$, $\tilde{\mathbf{f}}$, (5.12)-(5.13) of the agent-independent relaxed PEP formulation.

All agents are born free and equal
in dignity and rights.

6

Exploiting agents symmetries in PEP

We show that, in many settings, the worst-case performance of a distributed optimization algorithm is independent of the number of agents in the system, and can thus be computed in the fundamental case with just two agents. This result relies on a novel compact PEP formulation that systematically exploits symmetries in the worst-case performance computation.

In Chapter 4, we have observed that for many performance settings, the worst-case guarantees obtained with the spectral agent-dependent PEP formulation are independent of the number of agents, while the PEP problems are not. This motivates us to find compact ways of formulating the agent-dependent PEP. We have performed a first attempt in Chapter 5, where we built a relaxation of the problem whose size is independent of the number of agents n. In particular, the relaxation does not exploit the separability of the objective function F^{S_n} in (2.4). While the relaxed PEP formulation gives good worst-case bounds, it remains an open question whether it is equivalent to the agent-dependent formulation and how to interpret the resulting worst-case solution for the decentralized problem.

In this chapter, we provide an intuitive and systematic way of exploiting the agent symmetries in the Performance Estimation Problem (PEP) to make it compact, i.e., with a size independent of the number of agents. In addition, we characterize the situations where the worst-case performance of a decentralized algorithm is independent of the number of agents.

The chapter is organized as follows:

- In Section 6.1, we define an equivalence relation between agents in a PEP for decentralized optimization, and we leverage the convexity of the PEP to show that we can restrict it to solutions symmetrized over *equivalent agents*, without impacting its worst-case value.

- Section 6.2 focuses on the case where all the agents are equivalent, meaning none of them play a specific role in the algorithm or the performance evaluation. In that case, we show that the agent-dependent PEP can be written compactly, i.e., with an SDP whose size is independent of n. Moreover, we show that the worst-case value of this compact formulation is also independent of n in many common performance settings of distributed optimization which are scale-invariant. This result is particularly powerful since it allows determining situations where the performance analysis of a distributed algorithm can be reduced to the fundamental case with only two agents. We further leverage agent equivalence to draw general conclusions about the symmetry of worst-case local functions and iterates in decentralized algorithms.

- Section 6.3 generalizes the results from Section 6.2 to situations where there are several equivalence classes of agents in the PEP, leading to a *compact PEP formulation* whose size only depends on the number of equivalence classes r, and not directly on the total number of agents n. Indeed, while many performance estimation problems for decentralized optimization have all agents equivalent, there are more complex ones that involve multiple equivalence classes of agents, for example, when different groups of agents use different (uncoordinated) step-sizes, function classes, initial conditions, or even different algorithms. It can also happen that the performance measure focuses on a specific group of agents, e.g., the performance of the worst agent. Efficiently and accurately assessing distributed optimization algorithm performance in such advanced settings would enhance comprehensive analysis and deepen our understanding of their behavior.

- Section 6.4 compares the new *compact PEP formulation* with the spectral agent-dependent PEP formulation from Chapter 4. By construction, both formulations give the same worst-case guarantees.

This chapter is based on sections of the paper [CH24a].

6.1 Equivalence classes of agents in PEP

To exploit agent symmetries in the agent-dependent from Chapter 4, let us define an equivalence relation between agents, along with the corresponding equivalence classes. These will be used to build our new compact PEP formulation whose size depends only on the number of equivalence classes.

Let $\mathbf{f}$ and $G = P^T P$ be a solution of an agent-dependent PEP, described in (4.3)-(4.4). We can order their elements to regroup them depending on the agent to which they are associated:

$$\mathbf{f} = \begin{bmatrix} f_1^T & \dots & f_n^T \end{bmatrix}, \quad \text{where } f_i = \left[f_i^k\right]_{k \in I} \in \mathbb{R}^q \text{ is a vector with the } q \text{ functions values of agent } i, \tag{6.1}$$

Similarly, we write $P \in \mathbb{R}^{d \times np}$ as follows

$$P = \begin{bmatrix} P_1 & \dots & P_n \end{bmatrix} \quad \text{where } P_i \in \mathbb{R}^{d \times p} \text{ contains the } p \text{ vectors related to agent } i, \tag{6.2}$$

e.g., $P_i = \left[y_i^k \; x_i^k \; g_i^k\right]_{k \in I}$, and the Gram matrix $G \in \mathbb{R}^{np \times np}$ is thus defined as

$$G = P^T P = \begin{bmatrix} P_1^T P_1 & P_1^T P_2 & \dots \\ P_2^T P_1 & \ddots & \\ \vdots & & P_n^T P_n \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} & \dots \\ G_{21} & \ddots & \\ \vdots & & G_{nn} \end{bmatrix}. \tag{6.3}$$

Therefore, diagonal blocks G_{ii} are symmetric and off-diagonal blocks are such that $G_{ij} = G_{ji}^T$. We assume that each block P_i (and f_i) contains the same type and number of variables for each agent i. The variables common to all agents, such as x^*, are copied into each agent block P_i. This also covers the case where agents are heterogeneous and hold different types or numbers of variables since we can always add variables as columns of P_i (and f_i), even if agent i does not use them. Moreover, each agent block P_i (and f_i) has the same column order, so we can work with the same coefficient vectors for each agent. A coefficient vector for a variable x is denoted $e_x \in \mathbb{R}^p$, and contains linear coefficients selecting the correct combination of columns in $P_i \in \mathbb{R}^{d \times p}$ to obtain vector $x_i \in \mathbb{R}^d$, i.e., $P_i e_x = x_i$,

for any $i = 1, \dots, n$. This notation allows, for example, to write $x_i^T x_i$ as $x_i^T x_i = e_x^T P_i^T P_i e_x = e_x^T G_{ii} e_x$, for any agent i. Similarly, let $e_{f(x)}$ be a vector of coefficients selecting the elements in the vector f_i such that $e_{f(x)}^T f_i = f_i(x_i)$. These coefficient vectors will be used to explicit our new PEP formulations in Sections 6.2 and 6.3, as well as in Appendix 6.A.

Using this agent block notation for $\mathbf{f}$ and G, we can define permutations of agents in a PEP solution, leading to the definition of an equivalence relation between agents in performance estimation problems (Definition 6.2 below). For any feasible solution $(\mathbf{f}, G)$ of an agent-dependent PEP formulation (4.5), written in the form of (6.1) and (6.3), the solution obtained after permutation of agents $i, j \in \mathcal{V}$ in $\mathbf{f}$ and G is denoted $(\mathbf{f}_{\Pi_{ij}}, G_{\Pi_{ij}})$.

Example 6.1 (Permuted solutions). Let $(\mathbf{f}, G)$ be a feasible PEP solution written as

$$\mathbf{f} = \begin{bmatrix} f_1^T & f_2^T & \dots & f_n^T \end{bmatrix}$$
$$G = P^T P \qquad \text{with} \quad P = \begin{bmatrix} P_1 & P_2 & \dots & P_n \end{bmatrix}.$$

Agents 1 and 2 can be swapped to obtain a permuted solution $(\mathbf{f}_{\Pi_{12}}, G_{\Pi_{12}})$:

$$\mathbf{f}_{\Pi_{12}} = \begin{bmatrix} f_2^T & f_1^T & \dots & f_n^T \end{bmatrix}$$
$$G_{\Pi_{12}} = P_{\Pi_{12}}^T P_{\Pi_{12}} \qquad \text{with} \quad P_{\Pi_{12}} = \begin{bmatrix} P_2 & P_1 & \dots & P_n \end{bmatrix}.$$

Definition 6.2 (Equivalent agents)**.** Let $i, j \in \mathcal{V}$ be two distinct agents. We define an equivalence relation between the two agents $i \sim j$ if, for any feasible solution $(\mathbf{f}, G)$ (6.1)-(6.3) of an agent-dependent PEP formulation (4.5), the permuted solution $(\mathbf{f}_{\Pi_{ij}}, G_{\Pi_{ij}})$, with agent i and j swapped, is feasible for the PEP and provides the same performance value,

$$\mathcal{P}(\mathbf{f}, G) = \mathcal{P}(\mathbf{f}_{\Pi_{ij}}, G_{\Pi_{ij}}).$$

Intuitively, this means that equivalent agents have exactly the same role in the algorithm and the performance setting. By definition, this relation is reflexive, symmetric, and transitive, and thus corresponds well to an equivalence relation, which induces a partition into equivalence classes.

Definition 6.3 (Equivalence classes of agents)**.** Let $\mathcal{V}$ be a set of n agents. The equivalence relation from Definition 6.2 implies a partition $\mathcal{T}$ of set $\mathcal{V}$ into r equivalence classes (or subsets) of agents: $\mathcal{T} = \{\mathcal{V}_1, \dots, \mathcal{V}_r\}$. Each class only contains agents that are equivalent to each other. The size of each class $\mathcal{V}_u$ is denoted n_u $(u = 1, \dots, r)$. We also denote the equivalence class of a given agent i by $\mathcal{V}_{u_i}$.

Proposition 6.4 below proves the existence of a PEP solution for which equivalent agents have equal blocks in the solution. This will be the building block of our compact PEP formulations in Sections 6.2 and 6.3.

Proposition 6.4 (Existence of symmetric solution of a PEP)**.** *Let* $\mathbf{f} \in \mathbb{R}^{nq}$ *and* $G \in \mathbb{R}^{np \times np}$ *be any feasible solution of an agent-dependent PEP formulation for decentralized optimization* (4.5)*, and* $\mathcal{T} = \{\mathcal{V}_1, \dots, \mathcal{V}_r\}$ *be the partition of* $\mathcal{V}$ *induced by the equivalence relation from Definition 6.2. There is a symmetrized agent-class solution* $(\mathbf{f}^s_{\mathcal{T}}, G^s_{\mathcal{T}})$*, with equal blocks for equivalent agents, providing another valid solution for the PEP,*

$$\mathbf{f}^s_{\mathcal{T}} = \begin{bmatrix} (f^s_1)^T \dots (f^s_n)^T \end{bmatrix}, \qquad G^s_{\mathcal{T}} = \begin{bmatrix} G^s_{11} & G^s_{12} & \dots \\ G^s_{21} & \ddots & \\ \vdots & & G^s_{nn} \end{bmatrix},$$

$$\text{with } f^s_i = f^{u_i}_A := \frac{1}{n_{u_i}} \sum_{k \in \mathcal{V}_{u_i}} f_k, \qquad \text{for all } i \in \mathcal{V}, \tag{6.4}$$

$$G^s_{ij} = \begin{cases} G^{u_i}_A := \frac{1}{n_{u_i}} \sum_{k \in \mathcal{V}_{u_i}} G_{kk}, & \text{for } i = j, \\ G^{u_i}_B := \frac{1}{n_{u_i}(n_{u_i}-1)} \sum_{k \in \mathcal{V}_{u_i}} \sum_{\substack{l \in \mathcal{V}_{u_i} \\ l \neq k}} G_{kl}, & \text{for } j \neq i, j \in \mathcal{V}_{u_i}, \\ G^{u_i u_j}_E := \frac{1}{n_{u_i}} \sum_{k \in \mathcal{V}_{u_i}} G_{kj}, & \text{for } j \neq i, j \in \mathcal{V} \setminus \mathcal{V}_{u_i}. \end{cases} \tag{6.5}$$

Moreover, this symmetrized solution $(\mathbf{f}^s_{\mathcal{T}}, G^s_{\mathcal{T}})$ *provides the same performance value as* $(\mathbf{f}, G)$

$$\mathcal{P}(\mathbf{f}^s_{\mathcal{T}}, G^s_{\mathcal{T}}) = \mathcal{P}(\mathbf{f}, G).$$

Proof. By definition, any permutation of equivalent agents $\Pi \in \mathbb{R}^{n \times n}$ applied on a given PEP solution $\mathbf{f}$, G provides another valid PEP solution, with the same objective value:

$$\mathbf{f}_\Pi = \mathbf{f}(\Pi \otimes I_q), \quad P_\Pi = P(\Pi \otimes I_p), \quad G_\Pi = P_\Pi^T P_\Pi = (\Pi \otimes I_p)^T G (\Pi \otimes I_p).$$

A permutation of agents corresponds to a permutation of (sets of) columns and then the permutation matrix multiplies $\mathbf{f}$ and P on the right. The Kronecker products $\otimes$ are used to permute all the variables related to the same agent at the same time.

Since the PEP problem is linear in $\mathbf{f}$ and G, the combination of PEP solutions with the same objective value will give another PEP solution with the same objective value. We can thus construct a symmetrized PEP solu-

tion with the same objective value as $(\mathbf{f}, G)$ by averaging all the possible permuted solutions $(\mathbf{f}_\Pi, G_\Pi)$. Each permutation only involves agents from the same equivalence class, and therefore, the average applies to each class separately:

$$f_i^s = \frac{(n_{u_i}-1)!}{n_{u_i}!} \sum_{k \in \mathcal{V}_{u_i}} f_k = \frac{1}{n_{u_i}} \sum_{k \in \mathcal{V}_{u_i}} f_k = f_A^{u_i}, \qquad \text{for all } i \in \mathcal{V},$$

$$G_{ii}^s = \frac{(n_{u_i}-1)!}{n_{u_i}!} \sum_{k \in \mathcal{V}_{u_i}} G_{kk} = \frac{1}{n_{u_i}} \sum_{k \in \mathcal{V}_{u_i}} G_{kk} = G_A^{u_i}, \qquad \text{for all } i \in \mathcal{V},$$

$$G_{ij}^s = \frac{(n_{u_i}-2)!}{n_{u_i}!} \sum_{k \in \mathcal{V}_{u_i}} \sum_{\substack{l \in \mathcal{V}_{u_i} \\ l \neq k}} G_{kl} = \frac{1}{n_{u_i}(n_{u_i}-1)} \sum_{k \in \mathcal{V}_{u_i}} \sum_{\substack{l \in \mathcal{V}_{u_i} \\ l \neq k}} G_{kl} = G_B^{u_i}, \quad \text{for all } \substack{i \in \mathcal{V} \\ j \neq i \in \mathcal{V}_{u_i}},$$

$$G_{ij}^s = \frac{(n_{u_i}-1)!}{n_{u_i}!} \sum_{k \in \mathcal{V}_{u_i}} G_{kj} = \frac{1}{n_{u_i}} \sum_{k \in \mathcal{V}_{u_i}} G_{kj} = G_E^{u_i u_j}, \qquad \text{for all } \substack{i \in \mathcal{V} \\ j \neq i \in \mathcal{V} \setminus \mathcal{V}_{u_i}}.$$

□

Remark 6.5. The blocks composing $\mathbf{f}^s$ (6.4) are denoted by f_A^u. For the symmetrized Gram matrix G^s (6.5), we denote the three types of blocks composing it by G_A^u, G_B^u, and G_E^{uv}, for all equivalence classes $u, v = 1, \ldots, r$.

- Blocks $G_A^u \in \mathbb{R}^{p \times p}$ (for $u = 1, \ldots, r$) correspond to the blocks of the Gram matrix containing the scalar products between the variables of one agent from $\mathcal{V}_u$. These blocks lie on the diagonal of G^s and are symmetric by definition.

- Blocks $G_B^u \in \mathbb{R}^{p \times p}$ (for $u = 1, \ldots, r$) correspond to the blocks of the Gram matrix containing the scalar products between variables of two different agents from the same class $\mathcal{V}_u$. These blocks are symmetric because they can be written as a sum of symmetric matrices $\frac{1}{n_{u_i}(n_{u_i}-1)} \sum_{k,l \in \mathcal{V}_{u_i}, k>l} (G_{kl} + G_{lk}^T)$. Moreover, they are only defined for classes with at least 2 agents ($n_u \geq 2$).

- Blocks $G_E^{uv} \in \mathbb{R}^{p \times p}$ (for $u \neq v = 1, \ldots, r$) correspond to the blocks of the Gram matrix containing the scalar products between variables of two agents from different classes $\mathcal{V}_u$ and $\mathcal{V}_v$. These blocks are non-symmetric and are only defined when there are at least 2 different classes of agents ($r \geq 2$).

Therefore, we can solve the agent-dependent PEP problem, described in Chapter 4, restricted to symmetric solutions of the form of Proposition 6.4, without impacting the resulting worst-case value. The symmetry in the solutions allows working with a limited number of variables f_A^u, G_A^u, G_B^u, and G_E^{uv} in PEP, depending only on the (smaller) number of equivalent classes of agents r. This requires reformulating all the elements of the agent-dependent PEP (objective and constraints) in terms of these variables. Section 6.2 shows how to perform such a reformulation when all the agents are equivalent in the PEP, which occurs in many usual settings. This leads to a PEP formulation with only one equivalence class and hence whose size is independent of the number of agents. Section 6.3 generalizes reformulation techniques from Section 6.2 to a general number of equivalence classes.

Other symmetries that could be exploited In this chapter, we only consider symmetries of agents two by two, for which the permutations in the PEP do not impact its optimal value. This allows working with equivalence relations and classes of agents. However, there could also be symmetries involving more than two agents, such as cyclic symmetries, based on cyclic permutations of agents. We denote by $(\mathbf{f}_{\Pi_c}, G_{\Pi_c})$ the solution obtained after a cyclic permutation of agents $i_1, \dots, i_c \in \mathcal{V}$ in a feasible PEP solution $(\mathbf{f}, G)$. Specifically, this cyclic permutation is where $i_1 \to i_2$, $i_2 \to i_3$,…, $i_c \to i_1$.

Definition 6.6 (Cyclic agent symmetry in PEP)**.** We define a cyclic symmetry between the c agents $i_1 \dots, i_c$ in PEP if, for any feasible solution $(\mathbf{f}, G)$ (6.1)-(6.3) of an agent-dependent PEP formulation (4.5), the cyclic permuted solution $(\mathbf{f}_{\Pi_c}, G_{\Pi_c})$, is feasible for the PEP and provides the same performance value,

$$\mathcal{P}(\mathbf{f}, G) = \mathcal{P}(\mathbf{f}_{\Pi_c}, G_{\Pi_c}).$$

When $c = 2$, this amounts to Definition 6.2 of equivalent agents.

Such cyclic symmetries can also be used to partition the set of agents $\mathcal{V}$. If an agent is part of different cyclic symmetries, then they can be joined into one larger cyclic symmetry. Therefore, this type of symmetry can be exploited in the same manner as the equivalence classes in Proposition 6.4 to show the existence of symmetrized PEP solutions. However, we will not consider cyclic symmetries in this work, because they have limited applications and are less intuitive.

6.2 Agent-independent PEP when all agents are equivalent

This section focuses on cases where all agents are equivalent.

Assumption 6.7 (Equivalent Agents)**.** All the agents are equivalent in the Performance Estimation Problem applied to a distributed algorithm, in the sense of Definition 6.2. Therefore, there is only one equivalence class which is $\mathcal{V}$.

In other words, this assumption means that all the agents can be permuted in a PEP solution without impacting its worst-case value and thus no agent plays a specific role in the algorithm or its performance evaluation. This assumption is satisfied in many usual performance evaluation settings $\mathcal{S}_n$, for which all agents play an identical role in the algorithm and its performance evaluation. Under Assumption 6.7, the symmetrized solution from Proposition 6.4 only has a few different blocks that repeat.

Corollary 6.8 (Fully symmetric PEP solutions)**.** *When all the agents ($n \geq 2$) are equivalent, see Assumption 6.7, any PEP solution $(\mathbf{f}, G)$ can be fully symmetrized, without impacting its worst-case value:*

$$\mathbf{f}^s = \begin{bmatrix} f_A^T \dots f_A^T \end{bmatrix}, \qquad G^s = \begin{bmatrix} G_A & G_B & \dots \\ G_B & \ddots & \\ \vdots & & G_A \end{bmatrix},$$

$$\text{with} \quad f_A = \frac{1}{n}\sum_{i=1}^{n} f_i \ \in \mathbb{R}^q, \tag{6.6}$$

$$G_A = \frac{1}{n}\sum_{i=1}^{n} G_{ii} \ \in \mathbb{R}^{p\times p}, \quad G_B = \frac{1}{n(n-1)}\sum_{i=1}^{n}\sum_{j\neq i}^{n} G_{ij} \ \in \mathbb{R}^{p\times p}. \tag{6.7}$$

Proof. The results follow from applying Proposition 6.4 with $\mathcal{V}$ as the only equivalence class. Since there is only one equivalence class of agents, the Gram matrix only has two types of blocks, as explained in Remark 6.5. □

We consider the agent-dependent PEP formulation (4.1) from Chapter 4, with a spectral class of averaging matrix, and we constrain its solution to be fully symmetric, as expressed in (6.6) and (6.7), to obtain a compact PEP formulation with size independent of the number of agents. Then, we describe the implications of a fully symmetrized PEP solution on the worst-case local iterates and functions.

6.2.1 The PEP formulation restricted to fully symmetric solutions

When all agents are equivalent (Assumption 6.7), Corollary 6.8 shows that it does not impact the value of the worst-case, and we will see in Theorem 6.13 that it allows writing the PEP in a compact form since the performance measure and the constraints can be expressed only in terms of the smaller blocks f_A, G_A, and G_B that are repeating in $\mathbf{f}^s$ and G^s. Theorem 6.13 states sufficient conditions for which a PEP for distributed optimization can be made compact and for which it provides a worst-case value independent of the number of agents n. The theorem relies on the following definitions.

Definition 6.9 (Scale-invariant expression and constraint)**.** In a distributed system with n agents, we call a Gram-representable expression h scale-invariant if it can be written as a linear combination $h = \sum_j c_j h_j$, where coefficients c_j are independent of n and terms h_j are any of these three forms, for any vectors x_i and y_j assigned to agents i and j (including the case $x_i = y_j$):

(a) $\frac{1}{n}\sum_{i=1}^{n} f_i(x_i)$, the average of local function values,

(b) $\frac{1}{n}\sum_{i=1}^{n} x_i^T y_i$, the average of scalar products between two local variables of the same agent,

(c) $\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i^T y_j$, the average of the scalar products between local variables of any two agents.

A Gram representable constraint $h \leq D$, for $D \in \mathbb{R}$, is scale-invariant if the expression h is scale-invariant.

Remark 6.10. The name *scale-invariant* comes from the fact that, if we duplicate each agent of the system, with its local variables and its local function, then the value of a scale-invariant expression is unchanged. For example, the following expressions can be shown, possibly after algebraic manipulations, to be scale-invariant:

$$\frac{1}{n}\sum_{i=1}^{n}(f_i(\overline{x}) - f_i(x^*)),\ \frac{1}{n}\sum_{i=1}^{n}\|x_i - x^*\|^2,\ \frac{1}{n}\sum_{i=1}^{n}\|x_i - \overline{x}\|^2,\ \left\|\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(x_i)\right\|^2.$$

The variables common to all agents, such as $\overline{x}$ or x^*, can indeed be used in scale-invariant expressions because they are assumed to be copied into each agent set of variables:

$$\overline{x}_i = \overline{x} = \frac{1}{n}\sum_{j=1}^{n} x_j \quad \text{for all } i \in \mathcal{V} \qquad \text{and} \qquad x_i^* = x^* \quad \text{for all } i \in \mathcal{V}.$$

These constraints can then be written with scale-invariant expressions. See Appendix 6.A.1 for details.

Definition 6.11 (Single-agent expression and constraint)**.** In a distributed system with n agents, we call an expression h single-agent if it can be written as a linear combination $h = \sum_k c_k h_k(i)$, where coefficients c_k are independent of n and all terms $h_k(i)$ only involve the local variables and the local function of a single agent i. A constraint $h \leq D$, for $D \in \mathbb{R}$, is single-agent if the expression h is single-agent.

Remark 6.12. When all the agents are equivalent in the PEP, they all share the same single-agent constraints. Here are examples of single-agent constraints

$$\|x_i - x^*\|^2 \leq 1, \qquad \text{for all } i \in \mathcal{V}, \qquad\qquad \|\nabla f_i(x_i)\|^2 \leq 1, \qquad \text{for all } i \in \mathcal{V}.$$

We can now state Theorem 6.13 which characterizes the settings where the worst-case value $w(\mathcal{S}_n)$ can be computed (i) in a compact manner for all $n \geq 2$ and (ii) independently of the value of n. We remind that $w(\mathcal{S}_n = \{n, A, t, \mathcal{P}, \mathcal{F}, \mathcal{I}, \mathcal{W}\})$ denotes the worst-case performance of the execution of t iterations of distributed algorithm A with n agents, with respect to the performance criterion $\mathcal{P}$, valid for any starting point satisfying the set of initial conditions $\mathcal{I}$, any local functions in the set of functions $\mathcal{F}$ and any averaging matrix in the set of matrices $\mathcal{W}$.

Theorem 6.13 (Agents-independent worst-case performance)**.** *Let $n \geq 2$ and $\mathcal{W} = \mathcal{W}_{(\lambda^-, \lambda^+)}$ with $\lambda^-, \lambda^+ \in (-1, 1)$, $\lambda^- \leq \lambda^+$, defined in (4.10). We assume that the algorithm $A \in \mathcal{A}_D$, the performance measure $\mathcal{P}$, the initial conditions $\mathcal{I}$, and the interpolation constraints for $\mathcal{F}$ are linearly (or LMI) Gram-representable. If Assumption 6.7 holds, meaning that all the agents are equivalent in the PEP, then*

1. *The computation of the worst-case performance $w(\mathcal{S}_n)$ can be formulated as a compact SDP PEP, with $f_A \in \mathbb{R}^q$, $G_A \in \mathbb{R}^{p \times p}$ and $G_B \in \mathbb{R}^{p \times p}$ as variables, meaning that its size is independent of the number of agents n.*
2. *If, in addition, the performance criterion $\mathcal{P}$ is scale-invariant and the set of initial conditions $\mathcal{I}$ only contains single-agent constraints, applied to every agent $i \in \mathcal{V}$, and scale-invariant constraints, then the resulting SDP PEP problem and its worst-case value $w(\mathcal{S}_n)$ are fully independent of the number of agents n.*

Corollary 6.14. *Under the conditions stated in part 2 of Theorem 6.13, a general worst-case guarantee, uniformly valid for any $n \geq 2$ (including $n \to \infty$), can be obtained using $n = 2$:*

$$w(\mathcal{S}_n) = w(\mathcal{S}_2) \qquad \text{for all } n \geq 2.$$

Remark 6.15. The conditions of part 2 of Theorem 6.13, which guarantee agent-independent worst-case performance, are satisfied for many usual settings, as illustrated in Appendix 6.A.1.

To prove Theorem 6.13, we rely on Lemma 6.16 which gives a way of imposing the SDP condition $G^s \succeq 0$ solely based on its block matrices G_A and G_B.

Lemma 6.16. *Let $G^s \in \mathbb{R}^{np \times np}$ be a fully symmetrized Gram matrix, as defined in (6.7), for $n \geq 2$. The SDP constraint $G^s \succeq 0$ can be expressed with $G_A \in \mathbb{R}^{p \times p}$ and $G_B \in \mathbb{R}^{p \times p}$:*

$$G^s \succeq 0 \qquad \Leftrightarrow \qquad G_A + (n-1)G_B \succeq 0 \quad \text{and} \quad G_A - G_B \succeq 0. \tag{6.8}$$

Proof. We will use the fact that

$$G^s \succeq 0 \quad \Leftrightarrow \quad z^T G^s z \geq 0 \quad \text{for all } z \in \mathbb{R}^{np}.$$

Let us separate the vector $z \in \mathbb{R}^{np}$ into n sub-vectors z_i $(i = 1, \ldots, n)$ that can each be written as the sum of an average part $\bar{z} \in \mathbb{R}^p$ and a centered one $z_i^\perp$:

$$z_i = \bar{z} + z_i^\perp, \qquad \text{where } \bar{z} = \frac{1}{n}\sum_{i=1}^n z_i \text{ and } \sum_{i=1}^n z_i^\perp = 0.$$

Using this decomposition of z, together with the definition of G^s (6.7), we can express $z^T G^s z$ as follows

$$\begin{aligned}
z^T G^s z &= \begin{bmatrix} \bar{z} + z_1^\perp \\ \vdots \\ \bar{z} + z_n^\perp \end{bmatrix}^T \begin{bmatrix} G_A & G_B & \cdots \\ G_B & \ddots & \\ \vdots & & G_A \end{bmatrix} \begin{bmatrix} \bar{z} + z_1^\perp \\ \vdots \\ \bar{z} + z_n^\perp \end{bmatrix} \\
&= \sum_{i=1}^n \sum_{j=1}^n (\bar{z} + z_i^\perp)^T G_B (\bar{z} + z_j^\perp) + \sum_{i=1}^n (\bar{z} + z_i^\perp)^T (G_A - G_B)(\bar{z} + z_i^\perp), \\
&= n^2 \bar{z}^T (G_B)\bar{z} + n\bar{z}^T (G_A - G_B)\bar{z} + \sum_{i=1}^n (z_i^\perp)^T (G_A - G_B)(z_i^\perp), \\
&= \bar{z}^T (nG_A + n(n-1)G_B)\bar{z} + \sum_{i=1}^n (z_i^\perp)^T (G_A - G_B)(z_i^\perp).
\end{aligned}$$

We used the fact that $\sum_{i=1}^{n} z_i^{\perp} = 0$ to obtain the third equality. Therefore, $z^T G^s z \geq 0$ for all $z \in \mathbb{R}^{np}$ if, and only if, inequality (6.9) below holds for all $\bar{z}, z_i^{\perp} \in \mathbb{R}^p$ such that $\sum_{i=1}^{n} z_i^{\perp} = 0$:

$$\bar{z}^T(nG_A + n(n-1)G_B)\bar{z} + \sum_{i=1}^{n}(z_i^{\perp})^T(G_A - G_B)(z_i^{\perp}) \geq 0, \qquad (6.9)$$

Conditions $G_A + (n-1)G_B \succeq 0$ and $G_A - G_B \succeq 0$ are sufficient to ensure that (6.9) is satisfied and hence that $G^s \succeq 0$. To show the necessity of these conditions, let us consider different cases for $\bar{z}$ and $z_i^{\perp}$:

- When $z_i^{\perp} = 0$ for all i, then (6.9) simply imposes that $\bar{z}^T(nG_A + n(n-1)G_B)\bar{z} \geq 0$ for all $\bar{z} \in \mathbb{R}^p$, which is equivalent to $G_A + (n-1)G_B \succeq 0$.
- When $\bar{z} = 0$, $z_1^{\perp} = -z_2^{\perp} = s$, for an arbitrary $s \in \mathbb{R}^p$, and $z_i^{\perp} = 0$ for $i = 3, \ldots, n$, then (6.9) imposes that $s^T(G_A - G_B)s \geq 0$ for all $s \in \mathbb{R}^p$, which is equivalent to $G_A - G_B \succeq 0$.

□

Proof of Theorem 6.13 (part 1). As detailed in Section 4.3, the set of averaging matrices $\mathcal{W}_{(\lambda^-,\lambda^+)}$ has Gram-representable necessary interpolation constraints, see Corollary 4.8 and Proposition 4.9. Since $A, \mathcal{P}, \mathcal{I}$ and interpolation constraints for $\mathcal{F}$ are also Gram-representable, Proposition 3.7 guarantees that the computation of $w(n, A, t, \mathcal{P}, \mathcal{I}, \mathcal{F}, \mathcal{W}_{(\lambda^-,\lambda^+)})$ can be formulated as an agent-dependent SDP PEP problem (4.5). Moreover, since the agents are equivalent, we can restrict to fully symmetric solutions without impacting the worst-case value (see Corollary 6.8). Hence, all the PEP components can be written in terms of $\mathbf{f}^s$ and G^s, which are composed of blocks f_A, G_A, and G_B. Lemma 6.16 shows how the SDP constraint $G^s \succeq 0$, can be expressed in terms of G_A and G_B. The other elements of the PEP can directly be expressed in terms of f_A, G_A, and G_B, using the definition of $\mathbf{f}^s$ and G^s (6.6)-(6.7), as detailed in Appendix 6.A.1. Since all the PEP elements can be expressed in terms of blocks $f_A \in \mathbb{R}^q$, $G_A \in \mathbb{R}^{p\times p}$ and $G_B \in \mathbb{R}^{p\times p}$, the problem can be made smaller by considering them as the variables of the SDP PEP, instead of the full matrix $G^s \in \mathbb{R}^{np\times np}$ and vector $\mathbf{f}^s \in \mathbb{R}^{nq}$. The size of the problem is thus independent of the number of agents n, but n could still appear as a coefficient in the constraints or the objective of the PEP problem. □

It remains to prove the second part of Theorem 6.13, concerning the independence of n in worst-case performance. Lemma 6.16 makes the value of n appear as a coefficient, but we can define a change of variables to remove the dependency on n in the constraints (6.8). Let us defined the new matrix variables $G_C \in \mathbb{R}^{p\times p}$ as follows:

$$G_C = \frac{1}{n}(G_A + (n-1)G_B). \tag{6.10}$$

Moreover, we can also express G_C directly based on the blocks of the initial Gram matrix G (6.3):

$$G_C = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} G_{ij}. \tag{6.11}$$

While G_A contains the average scalar products between local variables of the same agent and G_B the average scalar products between local variables of different agents, this new block G_C contains the average scalar products between local variables of any two agents. By applying the change of variables (6.10) in Lemma 6.16, we obtain conditions that are independent of n:

Lemma 6.17 (Constraint $G^s \succeq 0$)**.** *Let $G^s \in \mathbb{R}^{np\times np}$ be a symmetric Gram matrix, as defined in (6.7). The SDP constraint $G^s \succeq 0$ can be expressed with $G_A \in \mathbb{R}^{p\times p}$ and $G_C \in \mathbb{R}^{p\times p}$:*

$$G^s \succeq 0 \qquad \Leftrightarrow \qquad G_C \succeq 0 \quad \textit{and} \quad G_A - G_C \succeq 0.$$

Proof. The result can be obtained by applying change of variables (6.10) in the constraints stated in (6.8). □

The new variable G_C allows writing many constraints and expressions without any dependence on n, as stated in the following lemma. This result will be used to prove the second part of Theorem 6.13.

Lemma 6.18. *In an SDP PEP for decentralized optimization (4.5), with equivalent agents (Assumption 6.7), any scale-invariant Gram-representable expression can be written independently of n, using variables f_A, G_A, and G_C.*

Proof. By Corollary 6.8, since all the agents are equivalent, we can restrict the PEP to fully symmetric solution $(\mathbf{f}^s, G^s)$ from (6.6)-(6.7). Since the expression is scale-invariant and Gram-representable (Definition 6.9), it can only combine three types of terms, with coefficients independent of n,

(a) The average of function values

$$\frac{1}{n}\sum_{i=1}^{n} f_i(x_i) = \frac{1}{n}\sum_{i=1}^{n} e_{f(x)}^T f_i = e_{f(x)}^T f_A, \tag{6.12}$$

where $e_{f(x)}$ is the vector of coefficients such that $e_{f(x)}^T f_i = f_i(x_i)$, with f_i defined in (6.1). The second equality holds because $f_i = f_A$ for all i, by definition of the agent-symmetric $\mathbf{f}^s$ (6.6).

(b) The average of scalar products between two variables related to the same agent i:

$$\frac{1}{n}\sum_{i=1}^{n} x_i^T y_i = \frac{1}{n}\sum_{i=1}^{n} e_x^T G_{ii} e_y = e_x^T G_A e_y, \tag{6.13}$$

where e_x (resp. e_y) is the vector of coefficients such that $P_i e_x = x_i$ (resp. $P_i e_y = y_i$), with P_i defined in (6.2). The second equality holds because $G_{ii} = G_A$, by definition of the agent-symmetric G^s (6.7).

(c) The average over the n^2 pairs of agents of the scalar products between two variables each related to any agent:

$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i^T y_j = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} e_x^T G_{ij} e_y = e_x^T G_C e_y, \tag{6.14}$$

where the last equality follows from the definition of G_C (6.11).

We have shown that the three types of terms (a), (b), (c) can be written independently of n with f_A, G_A, and G_C, and so can any Gram-representable and scale-invariant expression. □

We are now ready to prove the second part of Theorem 6.13, about the worst-case value being independent of the number of agents.

Proof of Theorem 6.13 (part 2). Under the conditions stated in the theorem (part 2), we will show that an agent-dependent SDP-PEP problem (4.5) can be written, independently of n, with f_A, G_A, and G_C as variables. When all the agents are equivalent in the PEP, we can restrict to fully symmetric solutions (Corollary 6.8). In that case, Lemma 6.17 shows that SDP constraints $G^s \succeq 0$ can be expressed with G_A and G_C, independently of n. The proof first focuses on single-agent constraints, by showing they can be

expressed independently of n, and detailing in which part of the PEP we find them. Then, we show that all the other PEP elements can be expressed with Gram-representable scale-invariant expressions (by definition or by assumption), which can be written independently of n using Lemma 6.18.

Let us prove that, in the symmetrized PEP, any single-agent constraint (Definition 6.11) can be written with f_A and G_A, independently of n. Since the PEP components are all Gram-representable, any of its single-agent constraints for agent i linearly combines local function values $f_i(x_i)$ and scalar products of local variables $x_i^T y_i$, with coefficients that are independent of n. Here, x_i and y_i denote any local vector held by agent i. By definition of the symmetric function values vector $\mathbf{f}^s$ (6.6), a local function value term can be expressed with f_A as

$$f_i(x_i) = e_{f(x)}^T f_i = e_{f(x)}^T f_A, \qquad \text{for any } i = 1, \dots, n. \tag{6.15}$$

By definition of the symmetric Gram G^s (6.7), a local scalar product term can be expressed with G_A as

$$x_i^T y_i = e_x^T G_{ii} e_y = e_x^T G_A e_y \qquad \text{for any } i = 1, \dots, n, \tag{6.16}$$

We have shown that the two types of terms (6.15) and (6.16) can be written independently of n with f_A, G_A, and so can any Gram-representable single-agent constraint. The resulting constraint, expressed with f_A and G_A, is valid for any agent $i \in \mathcal{V}$. This is consistent with the equivalent agents assumption, which requires that each single-agent constraint is applied similarly to every agent $i \in \mathcal{V}$. The reformulation of single-agent constraints, independently of n, applies to the interpolation constraints for $\mathcal{F}$, the algorithm constraints for $A \in \mathcal{A}_D$ (except consensus), and part of the initial constraints $\mathcal{I}$.

We now show that the other PEP components, see e.g., (4.1), can also be expressed independently of n. When agents are equivalent, by Lemma 6.18, any Gram-representable and scale-invariant expression can be expressed with f_A, G_A, and G_C, independently of n. As explained in the proof of part 1 of the theorem, all the PEP components are Gram-representable. Therefore, we will simply prove that all the PEP components that cannot be expressed as single-agent constraints, treated above, can be written with scale-invariant expressions. The performance measure $\mathcal{P}$ and the rest of the initial conditions $\mathcal{I}$ are scale-invariant, by assumption of the theorem statement. The optimality condition for (2.1), can be written in a Gram-

representable manner as

$$\left\| \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x^*) \right\|^2 = 0,$$

which is well scale-invariant thanks to the $1/n$ factor. Finally, the interpolation constraints for consensus steps with averaging matrices from $\mathcal{W}_{(\lambda^-,\lambda^+)}$ are given by (4.23), (4.24) and (4.25), from Corollary 4.8. In these constraints, notations hide sums over the agents and can be written with Gram-representable and scale-invariant expressions as

$$\begin{aligned}
&\overline{X} = \overline{Y} \;\Leftrightarrow\; \left(\frac{1}{n}\sum_{i=1}^{n}(X_i - Y_i)\right)^T \left(\frac{1}{n}\sum_{j=1}^{n}(X_j - Y_j)\right) = 0,\\
&(Y_\perp - \lambda^- X_\perp)^T (Y_\perp - \lambda^+ X_\perp) \preceq 0\\
&\qquad\Leftrightarrow\quad \frac{1}{n}\sum_{i=1}^{n} \big((Y_i - \overline{Y}) - \lambda^-(X_i - \overline{X})\big)^T \big((Y_i - \overline{Y}) - \lambda^+(X_i - \overline{X})\big) \preceq 0,\\
&X_\perp^T Y_\perp - Y_\perp^T X_\perp = 0\\
&\qquad\Leftrightarrow\quad \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^T(Y_i - \overline{Y}) - \frac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y})^T(X_i - \overline{X}) = 0,
\end{aligned}$$

where matrices $X_i, Y_i \in \mathbb{R}^{d\times t}$ contain the different consensus iterates $x_i^k, y_i^k \in \mathbb{R}^d$ $(k = 1,\dots,t)$ of agent i as columns. The two last constraints have been multiplied by $1/n$ to obtain scale-invariant expressions. This does not alter these constraints that have left-hand sides equal to zero.

In the end, all the PEP components can well be expressed independently of n, with $f_A \in \mathbb{R}^q$, $G_A \in \mathbb{R}^{p\times p}$ and $G_C \in \mathbb{R}^{p\times p}$. We refer to Appendix 6.A.1 for explicit expressions of different possible PEP constraints and objectives. Therefore, the SDP PEP can be made smaller and fully independent of the number of agents n, by considering these blocks as variables of the problem, instead of the full matrix $G^s \in \mathbb{R}^{np\times np}$ and vector $\mathbf{f}^s \in \mathbb{R}^{nq}$. □

6.2.2 Impact of agent symmetry on the worst-case solution

Proposition 6.19. *If all the agents are equivalent in the agent-dependent PEP (Assumption 6.7), then there is a worst-case instance for which,*

(i) the worst-case sequences of iterates of the agents $x_i^0, \dots, x_i^t \in \mathbb{R}^d$ ($i = 1, \dots, n$) are unitary transformations, i.e., rotations or reflections, of each other:

$$x_i^k = R_i x_1^k \quad \text{with } R_i \in \mathbb{R}^{d\times d} \text{ such that } R_i^T R_i = I, \quad \text{for all } i \in \mathcal{V} \text{ and all } k.$$

(ii) the worst-case local functions have equal values at their own iterates $f_i(x_i^k) = f_j(x_j^k)$ (for all $i, j \in \mathcal{V}$ and all k), and can be chosen identical up to a unitary transformation of variables:

$$f_i(x) = f_1(R_i^T x) \qquad \text{for all } i \in \mathcal{V}.$$

Remark 6.20. Proposition 6.19 is stated for the worst-case iterates, but is also valid for any linear combinations of columns of P_i (6.2). Indeed, as detailed in the proof below, we have $P_i = R_i P_1$, with $R_i \in \mathbb{R}^{d\times d}$ such that $R_i^T R_i = I$, for all $i = 1, \dots, n$. Moreover, the result is also valid for more than one agent at a time and can be written for any groups of agents, e.g., for groups of two agents, we have:

$$\begin{bmatrix} P_i & P_j \end{bmatrix} = \tilde{R}_{ij} [P_1 \; P_2] \quad \text{with } \tilde{R}_{ij} \in \mathbb{R}^{d\times d} \text{ such that } \tilde{R}_{ij}^T \tilde{R}_{ij} = I, \quad \text{for all } i, j \in \mathcal{V}.$$

This holds by definition of the fully symmetrized Gram matrix G^s (6.7) which has equal diagonal blocks: $G_{ii} = G_A$ for all $i \in \mathcal{V}$, but also equal off-diagonal blocks: $G_{ij} = G_B$ for $i \neq j \in \mathcal{V}$.

Proof of Proposition 6.19. By Corollary 6.8, when the agents are all equivalent, there is a worst-case solution of the agent-dependent PEP, which is fully symmetric: $(\mathbf{f}, G) = (\mathbf{f}^s, G^s)$. When the solution is agent-symmetric, by definition of G^s (6.7), we have

$$G_{ii} = P_i^T P_i = P_j^T P_j = G_{jj} \qquad \text{for all } i, j \in \mathcal{V},$$

which imposes that matrices of agent variables P_i are all unitary transformations of each other. Without loss of generality, we can express all the P_i depending on P_1:

$$P_i = R_i P_1 \quad \text{with } R_i^T R_i = I, \qquad \text{for all } i \in \mathcal{V}.$$

Matrices R_i are not especially unique. This unitary transformation relation is valid for any column or combination of columns of P_i and P_1, so we have it for iterates and gradient vectors of the agents:

$$x_i^k = R_i x_1^k \quad g_i^k = R_i g_1^k \quad \text{with } R_i^T R_i = I, \quad \text{for all } i \in \mathcal{V} \text{ and all } k. \tag{6.17}$$

For the local functions, by definition of $\mathbf{f}^s$ (6.6), we have, for all $i, j \in \mathcal{V}$,

$$f_i = f_j \quad \text{where } f_i \in \mathbb{R}^q \text{ is a vector with the functions values related to agent } i. \tag{6.18}$$

In particular, we have well that $f_i(x_i^k) = f_j(x_j^k)$, for all $i, j \in \mathcal{V}$ and all k. Let f_1 be a function interpolating the set of triplets $\{x_1^k, g_1^k, f_1^k\}_{k=0,\dots,t}$. We can choose f_i, the local function of agent i, as follows:

$$f_i(x) = f_1(R_i^T x).$$

Indeed, f_i interpolates well the set of triplets $\{x_i^k, g_i^k, f_i^k\}_{k=0,\dots,t}$ since (6.17) and (6.18) guarantee that $R_i^T x_i^k = x_1^k$, $R_i^T g_i^k = g_1^k$ and $f_i^k = f_1^k$, which are well interpolated by f_1 by definition. □

6.3 Compact PEP with multiple equivalence classes of agents

While the agents are all equivalent (Assumption 6.7) for many common performance evaluation settings, there are advanced settings for which there are several equivalence classes of agents, see Definition 6.3. For example, when different groups of agents use different (uncoordinated) step-sizes, function classes, initial conditions, or even different algorithms. It can also happen that the performance measure focuses on a specific group of agents, e.g., the performance of the worst agent. The ability to efficiently and accurately evaluate the performance of distributed optimization algorithms in such advanced settings would enable a more comprehensive analysis and deeper understanding of algorithms performance, as demonstrated for the EXTRA algorithm in Chapter 8.

In this section, we exploit symmetries in different equivalence classes of agents in PEP, revealed in Proposition 6.4, to write agent-dependent PEPs in a compact form whose size only depends on the number of equivalence

classes r, and not on the total number of agents n. By Proposition 6.4, we can solve the agent-dependent PEP problem, described in Chapter 4, restricted to symmetric agent-class solutions $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$, without impacting the worst-case value. These symmetric solutions $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$, defined in (6.4) and (6.5), depend on the partition $\mathcal{T}$ of the set of agents $\mathcal{V}$ into equivalence classes: $\mathcal{T} = \{\mathcal{V}_1, \dots, \mathcal{V}_r\}$, see Definition 6.3. A symmetrized solution $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$ has equal blocks for equivalent agents. In this section, we order the agents by equivalence classes in the symmetrized solution $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$ (6.4)-(6.5) so that we can write it as

$$\mathbf{f}_{\mathcal{T}}^s = \begin{bmatrix} \mathbf{f}_{\mathcal{V}_1} & \dots & \mathbf{f}_{\mathcal{V}_r} \end{bmatrix}^T, \qquad \text{with } \mathbf{f}_{\mathcal{V}_u} = \mathbf{1}_{n_u} \otimes f_A^u, \tag{6.19}$$

$$G_{\mathcal{T}}^s = \begin{bmatrix} G_{\mathcal{V}_1}^s & G_{\mathcal{V}_1\mathcal{V}_2}^s & \dots & G_{\mathcal{V}_1\mathcal{V}_r}^s \\ G_{\mathcal{V}_2\mathcal{V}_1}^s & G_{\mathcal{V}_2}^s & & \vdots \\ \vdots & & \ddots & \\ G_{\mathcal{V}_r\mathcal{V}_1}^s & \dots & & G_{\mathcal{V}_r}^s \end{bmatrix}, \tag{6.20}$$

$$\text{with} \quad G_{\mathcal{V}_u}^s = \begin{cases} G_A^u & \text{if } n_u = 1, \\ \mathbf{1}_{n_u}\mathbf{1}_{n_u}^T \otimes G_B^u + (I_{n_u} \otimes (G_A^u - G_B^u)) & \text{if } n_u \geq 2, \end{cases}$$

$$G_{\mathcal{V}_u\mathcal{V}_v}^s = \mathbf{1}_{n_u}\mathbf{1}_{n_v}^T \otimes G_E^{uv},$$

and where $f_A^u \in \mathbb{R}^q$, $G_A^u, G_B^u, G_E^{uv} \in \mathbb{R}^{p\times p}$ for $u, v = 1, \dots, r$ are the blocks repeating in $\mathbf{f}_{\mathcal{T}}^s$ and $G_{\mathcal{T}}^s$, which have been defined in Section 6.1 (see Remark 6.5).

Remark 6.21 (Limit cases for r).

- $r = n$: When no agents are equivalent, the partition $\mathcal{T}$ contains $r = n$ equivalence classes of size 1, and then the symmetrized agent-class solution $\mathbf{f}_{\mathcal{T}}^s$, $G_{\mathcal{T}}^s$, defined in Proposition 6.4, is identical to the agent-dependent PEP solution $\mathbf{f}$, G:

  $$\mathbf{f}_{\mathcal{T}}^s = \mathbf{f}, \qquad \text{and} \qquad G_{\mathcal{T}}^s = G.$$

  Therefore, the resulting PEP formulation is equivalent to the agent-dependent PEP from Chapter 4 and is not compact as its size is scaling with $r = n$.

- $r = 1$: When all the agents are equivalent, the partition $\mathcal{T}$ contains $r = 1$ equivalence class of size n, and thus, the symmetrized agent-

class solution $\mathbf{f}_{\mathcal{T}}^s$, $G_{\mathcal{T}}^s$ defined in Proposition 6.4 is identical to the fully symmetrized solution $\mathbf{f}^s$, G^s defined in Corollary 6.8:

$$\mathbf{f}_{\mathcal{T}}^s = \mathbf{f}^s, \qquad \text{and} \qquad G_{\mathcal{T}}^s = G^s.$$

Therefore, the resulting compact PEP formulation is identical to the one presented in Section 6.2.

6.3.1 The PEP formulation restricted to symmetric agent-class solutions

Based on the results from Proposition 6.4, the definitions of the symmetric PEP solutions $\mathbf{f}_{\mathcal{T}}^s$ (6.19) and $G_{\mathcal{T}}^s$ (6.20), we now show when and how an agent-dependent PEP can be expressed in a compact form, using only the blocks f_A^u, G_A^u, G_B^u, and G_E^{uv} $(u, v = 1, \ldots, r)$ as variables. This will enable efficient computing of $w(\mathcal{S}_n)$ for all $n \geq 2$, even when all the agents are not equivalent. We remind that $w(\mathcal{S}_n = \{n, A, t, \mathcal{P}, \mathcal{F}_u, \mathcal{I}, \mathcal{W}\})$ denotes the worst-case performance of the execution of t iterations of distributed algorithm A with n agents, with respect to the performance criterion $\mathcal{P}$, valid for any starting point satisfying the set of initial conditions $\mathcal{I}$, any local functions being each in a given set of function $f_i \in \mathcal{F}_{u_i}$ and any averaging matrix in the set of matrices $\mathcal{W}$.

Theorem 6.22 (Agent-class worst-case performance)**.** *Let $\mathcal{V}$ be a set of $n \geq 2$ agents, $\mathcal{T}$ be a partition of $\mathcal{V}$ into $r \leq n$ equivalence classes of agents, as defined in Definition 6.2, and $\mathcal{W} = \mathcal{W}_{(\lambda^-, \lambda^+)}$ with $\lambda^-, \lambda^+ \in (-1, 1)$, $\lambda^- \leq \lambda^+$, defined in (4.10).*
If the algorithm $A \in \mathcal{A}_D$, the performance measure $\mathcal{P}$, the initial conditions $\mathcal{I}$, and the interpolation constraints for each $\mathcal{F}_u$ $(u = 1, \ldots, r)$ are linearly (or LMI) Gram representable, then the computation of $w(\mathcal{S}_n)$ can be formulated as a compact SDP PEP, with f_A^u, G_A^u, G_B^u, and G_E^{uv} $(u, v = 1 \ldots, r)$ as variables, meaning that its size only depends on the number of agent equivalence classes r but not directly on the total number of agents n.

Remark 6.23. The blocks G_B^u contain scalar products between variables of different, but equivalent, agents. Therefore, G_B^u does not exist for equivalence classes with only one agent $n_u = 1$. The block G_E^{uv} contains scalar products between variables of different agents that are not equivalent. Therefore, this block does not exist when $r = 1$, i.e., all the agents are equivalent. The other blocks f_A^u and G_A^u, related to only one agent, always exist.

To prove this theorem, we rely on the following lemma which reformulates the SDP condition $G^s \succeq 0$ with the blocks G_A^u, G_B^u, and G_E^{uv}, and generalizes Lemma 6.16.

Lemma 6.24. *Let $G_{\mathcal{T}}^s \in \mathbb{R}^{np\times np}$ be a symmetrized agent-class Gram matrix, as defined in (6.20). The SDP constraint $G_{\mathcal{T}}^s \succeq 0$ can be expressed with $G_A^u \in \mathbb{R}^{p\times p}$, $G_B^u \in \mathbb{R}^{p\times p}$ and $G_E^{uv} \in \mathbb{R}^{p\times p}$, $u, v = 1\dots, r$:*

$$G_{\mathcal{T}}^s \succeq 0 \quad \Leftrightarrow \quad \begin{cases} G_A^u \succeq 0 & \text{for } u \text{ with } n_u = 1, \\ G_A^u - G_B^u \succeq 0 & \text{for } u \text{ with } n_u \geq 2, \end{cases} \quad \text{and} \quad H_{\mathcal{T}} \succeq 0, \tag{6.21}$$

where $H_{\mathcal{T}} \in \mathbb{R}^{rp\times rp}$ is a symmetric matrix composed of r^2 blocks of dimension $p \times p$:

$$H_{\mathcal{T}}^{uv} = \begin{cases} n_u G_A^u + n_u(n_u-1)G_B^u, & \text{when } u = v, \\ n_u n_v G_E^{uv}, & \text{when } u \neq v \end{cases}. \qquad u, v = 1, \dots, r. \tag{6.22}$$

Proof. We have that

$$G_{\mathcal{T}}^s \succeq 0 \quad \Leftrightarrow z^T G_{\mathcal{T}}^s z \geq 0 \quad \text{for all } z \in \mathbb{R}^{np}.$$

The vector $z \in \mathbb{R}^{np}$ can be divided into r sub-vectors of length $n_u p$ to match with the blocks of matrix $G_{\mathcal{T}}^s$ (6.20). We also decompose each sub-vector z_u in two terms : the average of z_u over the agents from the class $\mathcal{V}_u$, and the remaining part of $z_u^\perp$:

$$z_u = \mathbf{1}_{n_u} \otimes \bar{z}_u + z_u^\perp,$$

$$\text{with} \quad \bar{z}_u = \frac{1}{n_u}(\mathbf{1}_{n_u} \otimes I_p)^T z_u, \qquad \bar{z}_u \in \mathbb{R}^p,\ z_u^\perp \in \mathbb{R}^{n_u p} \text{ and } z_u \in \mathbb{R}^{n_u p}.$$

By definition, we have that $\frac{1}{n_u}(\mathbf{1}_{n_u} \otimes I_p)^T z_u^\perp = 0$ for all $j = 1, \dots, r$. We now compute $z^T G_{\mathcal{T}}^s z$ by summing over each block:

$$z^T G_{\mathcal{T}}^s z = \sum_{u=1}^r z_u^T G_{\mathcal{V}_u}^s z_u + \sum_{u=1}^r \sum_{v\neq u} z_u^T G_{\mathcal{V}_u\mathcal{V}_v}^s z_v, \tag{6.23}$$

where $G_{\mathcal{V}_u}^s$ and $G_{\mathcal{V}_u\mathcal{V}_v}^s$ are defined in (6.20). In the definition, we have two cases for $G_{\mathcal{V}_u}^s$, depending on the value of n_u. In the rest of the proof, we assume $n_u \geq 2$ and only work with $G_{\mathcal{V}_u}^s = \mathbf{1}_{n_u}\mathbf{1}_{n_u}^T \otimes G_B^u + (I_{n_u} \otimes (G_A^u - G_B^u))$. The special case for $n_u = 1$ ($G_{\mathcal{V}_u}^s = G_A^u$), can be recovered from the case $n_u \geq 2$, by defining $G_B^u = 0$ when $n_u = 1$. Here is the development for

each term of (6.23):

$$\begin{aligned} z_u^T G^s_{\mathcal{V}_u} z_u &= \left(\mathbf{1}_{n_u} \otimes \bar{z}_u + z_u^\perp\right)^T \left(\mathbf{1}_{n_u}\mathbf{1}_{n_u}^T \otimes G_B^u + (I_{n_u} \otimes (G_A^u - G_B^u))\right)\left(\mathbf{1}_{n_u} \otimes \bar{z}_u + z_u^\perp\right) \\ &= \bar{z}_u^T n_u^2 G_B^u \bar{z}_u + \bar{z}_u^T n_u (G_A^u - G_B^u)\bar{z}_u + (z_u^\perp)^T (I_{n_u} \otimes (G_A^u - G_B^u)) z_u^\perp, \\ &= \bar{z}_u^T (n_u G_A^u + n_u(n_u-1) G_B^u)\bar{z}_u + (z_u^\perp)^T (I_{n_u} \otimes (G_A^u - G_B^u)) z_u^\perp, \end{aligned}$$

$$\begin{aligned} z_u^T G^s_{\mathcal{V}_u\mathcal{V}_v} z_v &= \left(\mathbf{1}_{n_u} \otimes \bar{z}_u + z_u^\perp\right)^T \left(\mathbf{1}_{n_u}\mathbf{1}_{n_v}^T \otimes G_E^{uv}\right)\left(\mathbf{1}_{n_v} \otimes \bar{z}_v + z_v^\perp\right) \\ &= \bar{z}_u n_u n_v G_E^{uv} \bar{z}_v. \end{aligned}$$

Using these expressions, we can write (6.23) as a sum of quadratic forms

$$z^T G^s_{\mathcal{T}} z = \bar{z}^T H_{\mathcal{T}} \bar{z} + \sum_{u=1}^{r} (z_u^\perp)^T (I_{n_u} \otimes (G_A^u - G_B^u)) z_u^\perp,$$

where $\bar{z} \in \mathbb{R}^{rp}$ is a vector stacking the average vectors $\bar{z}_u$ of each equivalence class ($u = 1, \dots, r$), and $H_{\mathcal{T}} \in \mathbb{R}^{rp \times rp}$ is the matrix defined in (6.22). Therefore, the constraint $G^s_{\mathcal{T}} \succeq 0$ can be expressed as the following set of constraints, for all $\bar{z} \in \mathbb{R}^{rp}$ and all $z_u^\perp \in \mathbb{R}^{n_u p}$ such that $(\mathbf{1}_{n_u} \otimes I_p)^T z_u^\perp = 0$,

$$z^T G^s_{\mathcal{T}} z = \bar{z}^T H_{\mathcal{T}} \bar{z} + \sum_{u=1}^{r} (z_u^\perp)^T (I_{n_u} \otimes (G_A^u - G_B^u)) z_u^\perp \geq 0, \tag{6.24}$$

Conditions $H_{\mathcal{T}} \succeq 0$ and $G_A^u - G_B^u \succeq 0$ (for all $u = 1, \dots, r$) are sufficient to ensure that (6.24) is satisfied and hence that $G^s \succeq 0$. To show the necessity of these conditions, let us consider different cases for $\bar{z}$ and $z_u^\perp$:

- When $z_u^\perp = 0$ for all u, then (6.24) simply imposes that $\bar{z}^T H_{\mathcal{T}} \bar{z} \geq 0$ for all $\bar{z} \in \mathbb{R}^{rp}$, i.e., $H_{\mathcal{T}} \succeq 0$.
- When $\bar{z} = 0$, $z_u^\perp = 0$, for all u, except for an arbitrary $u = v$, then (6.24) imposes

  $$\sum_{i=1}^{n_v} (z_v^\perp(i))^T (G_A^v - G_B^v) z_v^\perp(i) \geq 0, \quad \text{for all } z_v^\perp \in \mathbb{R}^p \text{ s.t. } (\mathbf{1}_{n_v} \otimes I_p)^T z_v^\perp = 0,$$

  where $z_v^\perp(i) \in \mathbb{R}^p$, for $i = 1, \dots, n_v$, denotes the i^{th} part of vector $z_v^\perp \in \mathbb{R}^{n_v p}$. Since $n_v \geq 2$, we can choose $z_v^\perp(1) = -z_v^\perp(2) = s$, for an

arbitrary $s \in \mathbb{R}^p$, and $z_v^\perp(j) = 0$ for $j \geq 3$, so that the above equation becomes $s^T(G_A^v - G_B^v)s \geq 0$ for all $s \in \mathbb{R}^p$, which is equivalent to $G_A^v - G_B^v \succeq 0$. This is valid for an arbitrary $v = 1, \ldots, r$.

Therefore, by setting $G_B^u = 0$ when $n_u = 1$, we have well proved the equivalence (6.21) for the SDP condition $G_{\mathcal{T}}^s \succeq 0$. □

Proof of Theorem 6.22. As detailed in Section 4.3, the set of averaging matrices $\mathcal{W}_{(\lambda^-,\lambda^+)}$ has Gram-representable necessary interpolation constraints, see Corollary 4.8 and Proposition 4.9. Since $A, \mathcal{P}, \mathcal{I}$ and interpolation constraints for each $\mathcal{F}_u$ are also Gram-representable, Proposition 3.7 guarantees that the computation of $w(n, A, t, \mathcal{P}, \mathcal{I}, \mathcal{F}_u, \mathcal{W}_{(\lambda^-,\lambda^+)})$ can be formulated as an agent-dependent SDP PEP problem (4.5). Moreover, by Proposition 6.4, we can restrict the PEP to symmetric agent-class solutions of the form of $\mathbf{f}_{\mathcal{T}}^s$ (6.4) and $G_{\mathcal{T}}^s$ (6.5). The symmetric solutions $\mathbf{f}_{\mathcal{T}}^s$ and $G_{\mathcal{T}}^s$ are composed of blocks f_A^u, G_A^u, G_B^u, and G_E^{uv} $(u, v = 1 \ldots, r)$. Therefore, all the PEP elements can be expressed in terms of these smaller blocks. Lemma 6.24 shows how the SDP constraint $G_{\mathcal{T}}^s \succeq 0$, can be expressed in terms of G_A^u, G_B^u, and G_E^{uv}. The other elements of the PEP can directly be expressed in terms of f_A^u, G_A^u, G_B^u, and G_E^{uv} $(u, v = 1 \ldots, r)$, using the definition of $\mathbf{f}_{\mathcal{T}}^s$ (6.4) and $G_{\mathcal{T}}^s$ (6.7), as detailed in Appendix 6.A.2. Since all the PEP elements can be expressed in terms of blocks $f_A^u \in \mathbb{R}^q$, $G_A^u \in \mathbb{R}^{p\times p}$, $G_B^u \in \mathbb{R}^{p\times p}$ (when $n_u \geq 2$) and $G_E^{uv} \in \mathbb{R}^{p\times p}$ $(u, v = 1 \ldots, r)$, the problem can be made smaller by considering them as the variables of the SDP PEP, instead of the full matrix $G_{\mathcal{T}}^s \in \mathbb{R}^{np\times np}$ and vector $\mathbf{f}_{\mathcal{T}}^s \in \mathbb{R}^{nq}$. The size of the problem depends on the number r of equivalence classes of agents, and not directly on the total number of agents. □

Theorem 6.22 shows that the worst-case performance of a distributed optimization method can be computed with an SDP PEP whose size only depends on the number r of equivalence classes of agents, but not directly on the total number of agents n. Therefore, if the number of equivalence classes r is independent of n, so is the size of the problem. However, the problem can depend on n and n_u $(u = 1, \ldots, r)$ as parameters. Appendix 6.A.2 provides details to explicitly construct this compact PEP formulation.

6.3.2 Infinitely many agents

If the number r of equivalence classes of agents is independent of n, so is the size of the compact SDP PEP, and we can solve it efficiently for any

value of n. In particular, we can take the limit when n tend to ∞ in the problem, to compute the worst-case performance of an algorithm on an infinitely large network of agents

$$\lim_{n\to\infty} w(\mathcal{S}_n) \quad \text{where } \mathcal{S}_n = \{n, A, t, \mathcal{P}, \mathcal{I}, \mathcal{F}_u, \mathcal{W}_{(\lambda^-,\lambda^+)}\},$$

and to determine if this performance is bounded or not. In all the cases we have treated (see Chapter 8), we have observed that if the worst-case value for $n \to \infty$ is bounded, then it depends on the size proportions of each equivalence class $\rho_1, \ldots, \rho_r$:

$$\rho_u = \lim_{n\to\infty} \frac{n_u}{n}.$$

Future work could establish conditions, similar to those of Theorem 6.13 but applying to multiple equivalence classes of agents ($r \geq 1$), for which the worst-case performance depends on the size proportions of the classes $\rho_1, \ldots, \rho_r$, but not on the total number of agents n. This would extend the range of situations where we can guarantee that the worst-case performance is uniformly bounded for all n (including $n \to \infty$).

6.4 Comparison with the agent-dependent PEP formulation

Regardless of the number of equivalence classes, the compact PEP formulation, developed in this chapter, will always give the same worst-case value as the spectral agent-dependent PEP formulation from Chapter 4. Indeed, the compact PEP formulation searches only among symmetric agent-class solutions, which does not alter the worst-case value (see Proposition 6.4). This is precisely the reason why the compact formulation can work with a limited number of variables, namely $f_A^u \in \mathbb{R}^q$, $G_A^u \in \mathbb{R}^{p\times p}$, $G_B^u \in \mathbb{R}^{p\times p}$ (when $n_u \geq 2$) and $G_E^{uv} \in \mathbb{R}^{p\times p}$, for each equivalence classes $u, v = 1, \ldots, r$. Moreover, from a compact solution $(f_A^u, G_A^u, G_B^u, G_E^{uv})_{u,v=1,\ldots,r}$, if we know the number of agents in each equivalence class $n_1, \ldots, n_r$, then we can use relations (6.19)-(6.20) to construct a symmetric solution $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$ to the equivalent agent-dependent PEP. Worst-case local iterates and local functions of the different agents, associated with symmetric solutions, have specific symmetry properties, as shown in Proposition 6.19 for $r = 1$ equivalence class of agents.

We have implemented the compact PEP formulation in Matlab for different decentralized algorithms (including DGD, DIGing, and EXTRA). We used the YALMIP toolbox for modeling the problem and the Mosek solver for solving it. This allows us to verify that, as expected by the theory, the compact PEP formulation provides the same worst-case performance guarantees as the agent-dependent PEP formulation, when applied with the same performance evaluation settings $\mathcal{S}_n$.

Since the compact PEP formulation shares the same worst-case values as the agent-dependent PEP formulation, it also shares the same tightness analysis of the guarantees (see Chapter 4). In particular, both formulations compute guarantees valid for all averaging matrices from $\mathcal{W}_{(\lambda^-,\lambda^+)}$ (4.10), that are a priori non-tight but for which tightness can be verified a posteriori by recovering an instance of worst-averaging matrix in $\mathcal{W}_{(\lambda^-,\lambda^+)}$. For example, tightness has been verified a posteriori for DGD, DIGing, and EXTRA in different settings.

6.5 Summary and conclusion

In this chapter, we exploited agent symmetries to develop a compact SDP PEP formulation for computing the worst-case performance of decentralized optimization algorithms. By construction, the compact formulation provides the same worst-case performance guarantees as the agent-dependent PEP one from Chapter 4. When all the agents are equivalent, we characterized the situations where the performance is totally independent of the number of agents, which allows analyzing and computing the performance in the fundamental case with only two agents. Such situations include many common settings for the performance evaluation of distributed optimization algorithms. We also showed that in the worst-case scenario, when all agents are equivalent, their local sequences of iterates are rotations of each other and their local functions are identical up to a change of variables. Moreover, this new compact PEP formulation also allows analyzing advanced performance settings with several equivalence classes of agents, enabling tighter analysis and deeper understanding of the algorithm performance, as it will be demonstrated with EXTRA in Chapter 8. We believe that the different contributions of this chapter can significantly help the research in the field of distributed optimization by simplifying the analysis and understanding of the performance of distributed algorithms.

Appendix of Chapter 6

6.A Explicit expressions of the compact PEP components

This appendix provides explicit expressions for each component of the compact PEP formulation, to be able to use it easily in practice. We first focus on the case where all agents are equivalent in Section 6.A.1. Then, Section 6.A.2 details the case with multiple equivalence classes of agents.

6.A.1 When all agents are equivalent

Theorem 6.13 characterizes the settings in which we can formulate the performance computation of a distributed optimization algorithm as an SDP PEP which is totally independent of the number of agents n. In particular, we need all the agents to be equivalent (Assumption 6.7), so we can work with a fully symmetric solution $(\mathbf{f}^s, G^s)$ (Corollary 6.8), composed of small repeating blocks f_A, G_A, and G_B. The resulting agent-dependent PEP, expressed with blocks f_A, G_A, and G_B has a size independent of the number of agents n, but the value of n can still appear in the PEP and impact its solution, as any other parameter of the problem. Using suited changes of variables, we can obtain a PEP formulation that is fully independent of n, under the condition stated in part 2 of Theorem 6.13. In this appendix, we show how we can explicitly build this agent-independent PEP formulation for many usual settings. For that purpose, we pass through the main components of a PEP, that we have introduced in Chapter 4, and detail how they can be written with the blocks f_A, G_A, and G_B and which changes of variable get rid of n.

Firstly, the SDP condition $G^s \succeq 0$ can be expressed with G_A, and G_B using Lemma 6.16. Then, Lemma 6.17 makes it independent of n, by in-

troducing a new block $G_C = \frac{1}{n}(G_A + (n-1)G_B)$. Similarly to this result, the agent-independent formulation of other PEP components relies on the blocks G_A (6.7) and G_C (6.10) of the Gram matrices. We link these two matrices by defining *the difference block* G_D:

$$G_D = G_A - G_C. \tag{6.25}$$

We summarize below how these three blocks, related to the symmetrized Gram G^s by definition, can be expressed based on the blocks of any Gram matrix solution G (6.3):

$$G_A = \frac{1}{n}\sum_{i=1}^{n} G_{ii}, \quad G_C = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} G_{ij}, \quad \text{and} \quad G_D = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(G_{ii} - G_{ij}). \tag{6.26}$$

Using these three blocks, we can easily write many expressions in PEP, independently of n. However, only two of them should be variables of the resulting compact SDP PEP, as the third one could always be obtained using (6.25). In this appendix, we call agent-independently Gram-representable the expressions that can be written f_A, G_A, G_C, and G_D, without any dependence on the number of agents n.

Definition 6.25 (Agent-independently Gram-representable)**.** Let f_A be the block of the symmetrized function values vector, as defined in (6.6), G_A, G_C, and G_D the symmetric blocks defined in (6.26). We say that an expression, such as a constraint or an objective, is agent-independently Gram-representable if it can be expressed using a finite set of linear or LMI constraints or expressions involving only f_A, G_A, G_C, and G_D, without any dependence on the number of agents n.

In fact, an agent-independently Gram-representable expression can either be a single-agent constraint (Definition 6.11) or scale-invariant expression (Definition 6.9). The formulation of single-agent constraints into agent-independently Gram-representable constraints will rely on relations (6.15) and (6.16), involving f_A and G_A. The formulation of scale-invariant expressions into agent-independently Gram-representable expressions will rely on relations (6.12), (6.13), (6.14), involving respectively f_A, G_A, and G_C. We also define a useful relation involving G_D, which encapsulates a combination of relations (6.13) and (6.14). This new relation allows expressing the average over the n agents of the scalar products of

two centered variables related to the same agents:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^T(y_i-\overline{y}) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^Ty_i - x_i^Ty_j)$$
$$= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}e_x^T(G_{ii}-G_{ij})e_y = e_x^TG_De_y, \quad (6.27)$$

where x_i and y_j are any variables of agent i and j and $\overline{x} = \frac{1}{n}\sum_{j=1}^{n}x_i$ and $\overline{y} = \frac{1}{n}\sum_{j=1}^{n}y_i$ are their agent averages. The last equality in (6.27) follows from the definition of G_D (6.26). Moreover, $e_x, e_y \in \mathbb{R}^p$ denote coefficient vectors for variables x and y. As a reminder, a coefficient vector e_x contains linear coefficients selecting the correct combination of columns in $P_i \in \mathbb{R}^{d\times p}$ (6.2) to obtain vector $x_i \in \mathbb{R}^d$, i.e., $P_ie_x = x_i$, for any $i = 1,\dots,n$. This notation allows, for example, to write $x_i^Tx_i$ as $x_i^Tx_i = e_x^TP_i^TP_ie_x = e_x^TG_{ii}e_x$.

In what follows, we will exploit all these relations, to write all the components of a PEP, independently of n.
The following proposition treats the interpolation constraints of a PEP.

Proposition 6.26 (Function interpolation constraints)**.** *Let $\mathcal{F}$ be a set of functions for which there exist linearly Gram-representable interpolation constraints. In a PEP for a distributed optimization method, with fully symmetrized solutions $(\mathbf{f}^s, G^s)$ (6.6)-(6.7), the constraints $f_i \in \mathcal{F}$, for all $i \in \mathcal{V}$ can be expressed using a set of interpolation constraints that are agent-independently Gram-representable and that only involves f_A and G_A.*

Proof. Each constraint $f_i \in \mathcal{F}$ can be written with a set of Gram-representable interpolation constraints involving function values of agent i and scalar products between quantities (i.e., points and gradients) related to agent i. These are thus single-agent constraints that are all identical when we restrict to fully symmetric PEP solutions $(\mathbf{f}^s, G^s)$ (6.6)-(6.7). Hence, they can all be expressed with f_A and G_A using relations (6.15) and (6.16). □

Remark 6.27. For example, when $\mathcal{F}$ is the set of convex functions, we have,

$$\begin{aligned} f_i \in \mathcal{F} \quad &\Leftrightarrow \quad f_i^k \geq f_i^l + (g_i^l)^T(x_i^k - x_i^l) \quad \forall k,l \\ &\Leftrightarrow \quad e_{f^k}^Tf_A \geq e_{f^l}^Tf_A + e_{g^l}^TG_A(e_{x^k} - e_{x^l}), \end{aligned}$$

where $f_i^k = f_i(x_i^k)$, $g_i^l = \nabla f_i(x_i^l)$ and e_{f^k}, e_{f^l}, e_{g^l}, e_{x^k} and e_{x^l} are appropriate vectors of coefficients. The first line gives interpolation conditions for con-

vex functions, and the second on is obtained by applying relation (6.15) on $f_i(x_i^k)$ and $f_i(x_i^l)$ and relation (6.16) on the scalar product between g_i^l and $(x_i^k - x_i^l)$.

Algorithm description The class of methods $\mathcal{A}_D$ that we consider is defined in Definition 2.14 and may combine gradient evaluation, consensus steps, and linear combinations of variables. In the latter, each agent defines the same linear combination of its local variables. It should be the same combination for all the agents, otherwise, they are not equivalent, and Assumption 6.7 does not hold. All the variables in the combinations are related to the same agent and can therefore be squared and formulated in PEP using G_A, as in (6.13). Alternatively, we can reduce the number of variables in the blocks of the Gram matrix by using the linear combination to define a vector of coefficients in terms of the others, as shown in Proposition 6.28:

Proposition 6.28 (Linear combinations)**.** *Let $P_i = [y_i^1, \dots, y_i^p]$ be the p variables of agent i. We consider any linear combination of the variables, coordinated over all the agents $i \in \mathcal{V}$,*

$$z_i = \sum_{k=1}^{p} \beta_k y_i^k \qquad \text{for all } i, \tag{6.28}$$

where β_k are given coefficients and $y_i^k \in \mathbb{R}^d$ are any local variables from agent i. Such a combination step can be formulated in the PEP by defining the vector of coefficient e_z based on the vectors $e_{y^1}, \dots, e_{y^p}$.

$$e_z = \sum_{k=1}^{p} \beta_k e_{y^k}$$

This avoids adding z_i as a new column of P_i.

Proof. By definition, we have $z_i = P_i e_z$ and $y_i^k = P_i e_{y^k}$. Therefore, the linear equality (6.28) can be written as

$$P_i \left(e_z - \sum_{k=1}^{p} \beta_k e_{y^k} \right) = 0,$$

which allows defining e_z based on the other vectors of coefficients $e_z = \sum_{k=1}^{p} \beta_k e_{y^k}$, without adding a new column in P_i. □

Concerning the consensus steps, Theorem 6.13 applies to the set of averaging matrices $\mathcal{W}_{(\lambda^-,\lambda^+)}$, defined in (4.10) in Chapter 4. Necessary interpolation constraints for this set of matrices are given in (4.23), (4.24) and (4.25), from Corollary 4.8. Proposition 6.29 below shows that these interpolation constraints are agent-independently Gram-representable, by expressing them in terms of G_C and G_D. In principle, the compact PEP formulation could apply to a general set of averaging matrices $\mathcal{M}$ for which there exist Gram-representable interpolation constraints. Future work may include the description of other classes of averaging matrices, in particular, the extension to non-symmetric stochastic matrices with a bound on their singular value.

Proposition 6.29 (Averaging matrix interpolation constraints)**.** *The averaging matrix interpolation constraints* (4.23), (4.24) *and* (4.25)*, from Corollary 4.8, are agent-independently Gram-representable and can be expressed using G_C and $G_D \in \mathbb{R}^{p\times p}$ as:*

$$(e_X - e_Y)^T G_C (e_X - e_Y) = 0, \tag{6.29}$$

$$(e_Y - \lambda^- e_X)^T G_D (e_Y - \lambda^+ e_X) \preceq 0, \tag{6.30}$$

$$e_Y^T G_D e_X - e_X^T G_D e_Y = 0, \tag{6.31}$$

where $e_X, e_Y \in \mathbb{R}^p$ are matrices of coefficients such that $P_i e_X = X_i$, $P_i e_Y = Y_i$ with $X_i, Y_i \in \mathbb{R}^{d\times t}$ the matrices with iterates $x_i^k, y_i^k \in \mathbb{R}^d$ as columns.

Proof. Firstly, we express (4.23) as (6.29). Equation (4.23), when applied to the consensus subspace, means that the agent average is preserved, i.e., $\overline{X} = \overline{Y}$, which can be written with scalar products as:

$$\left(\frac{1}{n}\sum_{i=1}^n (X_i - Y_i)\right)^T \left(\frac{1}{n}\sum_{j=1}^n (X_j - Y_j)\right) = 0,$$

where matrices $X_i, Y_i \in \mathbb{R}^{d\times t}$ contain the different consensus iterates $x_i, y_i \in \mathbb{R}^d$ of agent i as columns. Using relation (6.14), applied to columns of $X_i - Y_i$ and $X_j - Y_j$, this equation can be written as

$$\frac{1}{n^2}\sum_{i=1}^n\sum_{j=1}^n (X_i - Y_i)^T (X_j - Y_j) = (e_X - e_Y)^T G_C (e_X - e_Y) = 0.$$

Secondly, we express (4.24) as (6.30). Equation (4.24) involve centered ma-

trices $X_\perp, Y_\perp \in \mathbb{R}^{nd\times t}$,

$$(Y_\perp - \lambda^- X_\perp)^T (Y_\perp - \lambda^+ X_\perp) \preceq 0, \tag{6.32}$$

We can write this product as a sum over the agents of products of centered variables, which then allows using relation (6.27) to express it with G_D. To this end, let us define $U_i = Y_i - \lambda^- X_i$, $V_i = Y_i - \lambda^+ X_i$, and $\overline{U}, \overline{V}$ their corresponding agent average: $\overline{U} = \frac{1}{n}\sum_{i=1}^n U_i$, $\overline{V} = \frac{1}{n}\sum_{i=1}^n V_i$. Equation (6.32) can be written as

$$\sum_{i=1}^n (U_i - \overline{U})^T (V_i - \overline{V}) \preceq 0.$$

This can be reformulated using relation (6.27) applied to columns of U_i and V_i and dividing the equation by n:

$$\frac{1}{n}\sum_{i=1}^n (U_i - \overline{U})^T (V_i - \overline{V}) = e_r^T G_D e_V = (e_Y - \lambda^- e_X)^T G_D (e_Y - \lambda^+ e_X) \preceq 0.$$

Finally, the symmetry condition (4.25) can be expressed as (6.31) using a similar technique

$$X_\perp^T Y_\perp - Y_\perp^T X_\perp = \sum_{i=1}^n (X_i - \overline{X})^T (Y_i - \overline{Y}) - \sum_{i=1}^n (Y_i - \overline{Y})^T (X_i - \overline{X}) = 0.$$

This can be reformulated using relation (6.27) applied to columns of X_i and Y_i and dividing the equation by n:

$$e_Y^T G_D e_X - e_X^T G_D e_Y = 0.$$

□

Points common to all agents (x^*, $\overline{x}$, etc) As explained in Section 6.1, to build our compact PEP formulation for distributed optimization, we divide the agent-dependent PEP solutions into blocks of variables related to each agent, see (6.1) and (6.2), recalled below

$$\mathbf{f} = \begin{bmatrix} f_1^T & \dots & f_n^T \end{bmatrix}, \quad \text{where } f_i = \left[f_i^k\right]_{k\in I} \in \mathbb{R}^q \text{ is a vector with the } q \text{ functions values of agent } i,$$

$$P = \begin{bmatrix} P_1 & \dots & P_n \end{bmatrix} \qquad \text{where } P_i \in \mathbb{R}^{d\times p} \text{ contains the } p \text{ vectors related to agent } i,$$

e.g., $P_i = \begin{bmatrix} y_i^k & x_i^k & g_i^k \end{bmatrix}_{k\in I}$, and the Gram matrix $G \in \mathbb{R}^{np\times np}$ is thus defined as

$$G = P^T P = \begin{bmatrix} P_1^T P_1 & P_1^T P_2 & \dots \\ P_2^T P_1 & \ddots & \\ \vdots & & P_n^T P_n \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} & \dots \\ G_{21} & \ddots & \\ \vdots & & G_{nn} \end{bmatrix}.$$

The variables common to all agents, such as x^* or $\overline{x}^t$, are copied into each agent block P_i, and their definitions are added as constraints of the problem, e.g.,

$$\overline{x}_i = \overline{x} = \frac{1}{n}\sum_{j=1}^{n} x_j \quad \text{for all } i \in \mathcal{V}, \tag{6.33}$$

$$\text{or} \qquad \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x_i^*) = 0, \quad \text{with } x_i^* = x_j^* = x^* \text{ for all } i, j \in \mathcal{V}. \tag{6.34}$$

Such definition constraints are agent-independently Gram-representable, as shown in Propositions 6.30 and 6.32 below. If used in the problem, each agent i also holds the variables for its local function and gradient values associated with this common point. For example, for x^*, we consider a new triplet (x_i^*, g_i^*, f_i^*) for each agent i, which is taken into account in the interpolation constraints of its local function. These interpolation constraints ensure that g_i^* and f_i^* correspond to a gradient vector and function value at x^* that are consistent with the given class of functions. Vectors x_i^* and g_i^* are columns of P_i and value f_i^* is an element of vector f_i.
To summarize, for any point x_c common to all the agents in the problem, we need two elements in the PEP:

(i) the definition constraints for x_c, e.g., (6.33) or (6.34),

(ii) the interpolation constraints for the new triplet (x_c, g_c, f_c), if g_c or f_c are used in the PEP.

The addition of a new triplet of points to consider in the interpolation constraints does not alter the result of Proposition 6.34 and they can still be written as agent-independently Gram-representable constraints with f_A and G_A. The definition constraints can be treated as any other constraint of the problem, e.g., the initial constraints, and these constraints can thus be formulated independently of n if they can be written with scale-invariant

or single-agent expressions (see Definition 6.9, Definition 6.11 and Theorem 6.13). Proposition 6.30 gives an agent-independent formulation for the definition constraint of the optimal solution x^* of the decentralized optimization problem (2.1).

Proposition 6.30 (Optimality constraint)**.** *Let x^* be an optimal point for the decentralized optimization problem (2.1). The definition constraints for the common variable x^* in a PEP restricted to fully symmetric solutions $(\mathbf{f}^s, G^s)$ (6.6)-(6.7), given by the system-level stationarity constraint*

$$\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(x_i^*) = 0, \quad \text{with } x_i^* = x_j^* = x^* \text{ for all } i,j \in \mathcal{V}, \tag{6.35}$$

is agent-independently Gram-representable and can be expressed as

$$e_{g^*}^T G_C e_{g^*} = 0 \quad \text{and} \quad e_{x^*}^T G_D e_{x^*} = 0,$$

where $e_{g^*}, e_{x^*} \in \mathbb{R}^p$ are vectors of coefficients such that $P_i e_{g^*} = g_i^* = \nabla f_i(x^*)$, and $P_i e_{x^*} = x_i^*$ for all $i = 1, \dots, n$.

Proof. The first constraint in (6.35) guarantees that the optimal point x_i^* held by each agent satisfies the system-level stationarity constraint $\frac{1}{n}\sum_{i=1}^n \nabla f_i(x_i^*) = 0$. Then, the second set of constraints imposes that the local optimal point x_i^* is the same for any agent i. Let g_i^* denote the gradient of function f_i at the optimal point x_i^*. We can formulate the stationarity constraint using only scalar products as

$$\left(\frac{1}{n}\sum_{i=1}^{n} g_i^*\right)^T \left(\frac{1}{n}\sum_{j=1}^{n} g_j^*\right) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(g_i^*)^T g_j^* = 0.$$

Using relation (6.14), applied to vectors g_i^*, g_j^*, we recover condition $e_{g^*}^T G_C e_{g^*} = 0$. For the second set of constraints, we can impose them all with one constraint involving only scalar products:

$$\frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^* - x_j^*)^T(x_i^* - x_j^*) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^*)^T x_i^* - (x_j^*)^T x_j^* = 0,$$

which can be written, using relation (6.27), as

$$\begin{aligned}\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^*)^T x_i^* - (x_i^*)^T x_j^* &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} e_{x^*}^T(G_{ii} - G_{ij})e_{x^*} \\ &= e_{x^*}^T G_D e_{x^*}.\end{aligned}$$

□

Remark 6.31. Without loss of generality, the optimal solution x^* of problem (2.1) can be set to $x^* = 0$. In that case, the constraint $x_i^* = x_j^*$, written as $e_{x^*}^T G_D e_{x^*} = 0$, is not needed to define x^* properly. Indeed, it is sufficient to say that the coefficient vector is zero: $e_{x^*} = 0$. This improves the numerical conditioning of the resulting SDP PEP.

Proposition 6.32 gives an agent-independent formulation for the definition constraint of the agent average iterate $\overline{x}^k$. To simplify notation, we omit the k index for iterations.

Proposition 6.32 (Agent average definition)**.** *We consider a PEP for distributed optimization, restricted to fully symmetric solutions* $(\mathbf{f}^s, G^s)$ (6.6)-(6.7)*. The definition constraints for* $\overline{x}$*,*

$$\overline{x}_i = \overline{x} = \frac{1}{n}\sum_{j=1}^{n} x_j \quad \text{for all } i \in \mathcal{V}, \tag{6.36}$$

are agent-independently Gram-representable, and can be written as

$$e_{\overline{x}}^T G_A e_{\overline{x}} + e_x^T G_C (e_x - 2e_{\overline{x}}) = 0,$$

where $e_{\overline{x}}, e_x \in \mathbb{R}^p$ *are vectors of coefficients such that* $P_i e_{\overline{x}} = \overline{x}_i$*, and* $P_i e_x = x_i$ *for all* $i = 1, \dots, n$*.*

Proof. We can formulate the set of constraints (6.36) with one constraint involving only scalar products of variables:

$$\frac{1}{n}\sum_{i=1}^{n}\left(\overline{x}_i^t - \frac{1}{n}\sum_{j=1}^{n} x_j^t\right)^T\left(\overline{x}_i^t - \frac{1}{n}\sum_{l=1}^{n} x_l^t\right) = 0.$$

This expression can be expanded to be written with G_A and G_C:

$$\frac{1}{n}\sum_{i=1}^{n}\overline{x}_i^T\overline{x}_i + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_j^T x_i - 2x_j^T\overline{x}_i) = e_{\overline{x}}^T G_A e_{\overline{x}} + e_x^T G_C(e_x - 2e_{\overline{x}}) = 0,$$

where we used relation (6.13) for the first term and (6.14) for the other two. □

Initial conditions and performance measures By combining terms of the form of (6.12), (6.13), (6.14), (6.15), (6.16), and (6.27), possibly involving common points defined above, it is possible to express many initial conditions and performance measures in terms of f_A, G_A, G_C, and G_D, without

dependence on n. Examples of usual initial conditions and their reformulation are given in Proposition 6.33 below. The proposition focuses on the initial conditions, but all the expressions involved in these constraints can also be adapted as a performance measure, using the last point x^t instead of the initial point x^0.

Proposition 6.33 (Initial conditions)**.** *Let $x^* \in \mathbb{R}^d$ denote the optimal solution of the distributed optimization problem (2.1), $x_i^0 \in \mathbb{R}^d$ the initial iterate of agent i, $\overline{x}^0$ the average initial iterate and $R \in \mathbb{R}$ a constant. Here is a list of initial conditions that are agent-independently Gram-representable when restricted to agent-symmetric solutions, together with their reformulations in terms of f_A, G_A, G_C, and G_D:*

Initial condition		*Reformulation*
$\frac{1}{n}\sum_{i=1}^n \|x_i^0 - x^*\|^2 \leq R^2,$		$(e_{x^0} - e_{x^*})^T G_A (e_{x^0} - e_{x^*}) \leq R^2,$
or $\|x_i^0 - x^*\|^2, \leq R^2, \quad \forall i \in \mathcal{V}$		
$\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(x_i^0)\|^2 \leq R^2,$		$e_{g^0}^T G_A e_{g^0} \leq R^2,$
or $\|\nabla f_i(x_i^0)\|^2, \leq R^2, \quad \forall i \in \mathcal{V}$		
$\frac{1}{n}\sum_{i=1}^n \|x_i^0 - \overline{x}^0\|^2 \leq R^2,$		$e_{x^0}^T G_D e_{x^0} \leq R^2,$
$x_i^0 - x_j^0 = 0, \quad \forall i,j \in \mathcal{V}$		$e_{x^0}^T G_D e_{x^0} = 0,$
$\frac{1}{n}\sum_{i=1}^n \left(f_i(\overline{x}^0) - f_i(x^*)\right) \leq R,$		$(e_{f(\overline{x}^0)} - e_{f(x^*)})^T f_A (e_{f(\overline{x}^0)} - e_{f(x^*)}) \leq R.$

where $e_{x^0}, e_{x^*}, e_{g^0} \in \mathbb{R}^p$ and $e_{f(\overline{x}^0)}, e_{f(x^*)} \in \mathbb{R}^q$ are vectors of coefficients such that $P_i e_{x^0} = x_i^0$, $P_i e_{x^*} = x^*$, $P_i e_{g^0} = \nabla f_i(x_i^0)$, $e_{f(\overline{x}^0)}^T f_i = f_i(\overline{x}^0)$ and $e_{f(x^*)}^T f_i = f_i(x^*)$, for all $i = 1, \ldots, n$.

Proof. To obtain the two first reformulations, we can use relation (6.13) applied respectively to vectors $x_i^0 - x^*$ and $\nabla f_i(x_i^0)$

$$\frac{1}{n}\sum_{i=1}^n \|x_i^0 - x^*\|^2 = \frac{1}{n}\sum_{i=1}^n (x_i^0 - x^*)^T (x_i^0 - x^*) = (e_{x^0} - e_{x^*})^T G_A (e_{x^0} - e_{x^*}),$$
$$\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(x_i^0)\|^2 = \frac{1}{n}\sum_{i=1}^n (\nabla f_i(x_i^0))^T (\nabla f_i(x_i^0)) = e_{g^0}^T G_A e_{g^0}.$$

When considering the alternative uniform conditions for all agents, we obtain the same reformulations using relation (6.16), applied respectively to vectors $x_i^0 - x^*$ and $\nabla f_i(x_i^0)$. For the third condition, we can directly use relation (6.27) to obtain it. Then, the fourth condition is equivalent to

$$\frac{1}{2n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^0 - x_j^0)^T(x_i^0 - x_j^0) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^0)^T x_i^0 - (x_i^0)^T x_j^0 = 0,$$

which can be written, using relation (6.27), as

$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^0)^T x_i^0 - (x_i^0)^T x_j^0 = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} e_{x^0}^T(G_{ii} - G_{ij})e_{x^0} = e_{x^0}^T G_D e_{x^0} = 0.$$

Finally, the fifth and last condition can be reformulated with f_A using relation (6.12) applied to $\frac{1}{n}\sum_{i=1}^{n} f_i(\overline{x}^0)$ and $\frac{1}{n}\sum_{i=1}^{n} f_i(x^*)$. □

6.A.2 With multiple equivalence classes of agents

Let $\mathcal{T}$ be a partition of the agent set $\mathcal{V}$ into r equivalence classes of agents (see Definition 6.3). Considering the PEP restricted to symmetrized agent-class solutions $G^s_{\mathcal{T}}$ and $\mathbf{f}^s_{\mathcal{T}}$ (6.19)-(6.20), we now detail how we can express all the PEP elements in terms of f^u_A, G^u_A, G^u_B, and G^{uv}_E $(u,v = 1\dots,r)$, to be able to use this blocks as variable of the compact SDP PEP formulation. For the SDP condition $G^s \succeq 0$, we can be express it with G^u_A, G^u_B, and G^{uv}_E using Lemma 6.24. For the other PEP elements, we need to adapt the reformulation techniques (6.12), (6.13), (6.14), (6.15), (6.16) and (6.27) to the situation where the agent set $\mathcal{V}$ is partitioned into r subsets of equivalent agents $\mathcal{T} = \{\mathcal{V}_1,\dots,\mathcal{V}_r\}$. The terms that we can be expressed in terms of f^u_A, G^u_A, G^u_B, and G^{uv}_E are the following (for any vectors x_i and y_j related to agents i and j, including the case $x_i = y_j$):

- The function value of a given agent $j \in \mathcal{V}_u$

$$f_j(x_j) = \frac{1}{n_u}\sum_{i\in\mathcal{V}_u} f_i(x_i) = e^T_{f(x)} f^u_A, \qquad \text{for any } j = 1,\dots,n, \tag{6.37}$$

  where the first equality holds because $f_i(x_i) = f_j(x_j)$ for any $i,j \in \mathcal{V}_u$, by definition of the symmetric agent-class solution $\mathbf{f}^s_{\mathcal{T}}$ (6.19). This implies that

$$\frac{1}{n}\sum_{i=1}^{n} f_i(x_i) = \frac{1}{n}\sum_{u=1}^{r}\sum_{j\in V_u}^{n} f_j(x_j) = \frac{1}{n}\sum_{u=1}^{r} n_u e^T_{f(x)} f^u_A = e^T_{f(x)} f_{A,\mathcal{T}}, \tag{6.38}$$

  where vector $f_{A,\mathcal{T}} \in \mathbb{R}^q$ is given by

$$f_{A,\mathcal{T}} = \frac{1}{n}\sum_{u=1}^{r} n_u f^u_A. \tag{6.39}$$

- The scalar product between two variables related to the same agent $i \in \mathcal{V}_u$:

$$x_i^T y_i = \frac{1}{n_u}\sum_{j\in V_u}^{n} x_j^T y_j = e_x^T G^u_A e_y \qquad \text{for any } i = 1,\dots,n, \tag{6.40}$$

  where the first equality holds because $x_i^T y_i = x_j^T y_j$ for any $i,j \in \mathcal{V}_u$, by definition of the symmetric agent-class solution $G^s_{\mathcal{T}}$ (6.20). This

implies that

$$\frac{1}{n}\sum_{i=1}^{n} x_i^T y_i = \frac{1}{n}\sum_{u=1}^{r}\sum_{j\in V_u}^{n} x_j^T y_j = \frac{1}{n}\sum_{u=1}^{r} n_u e_x^T G_A^u e_y = e_x^T (G_{A,\mathcal{T}}) e_y, \tag{6.41}$$

where matrix $G_{A,\mathcal{T}} \in \mathbb{R}^{p\times p}$ is given by

$$G_{A,\mathcal{T}} = \frac{1}{n}\sum_{u=1}^{r} n_u G_A^u. \tag{6.42}$$

- The average over the n^2 pairs of agents of the scalar products between two variables each related to any agent:

$$\begin{aligned}\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i^T y_j &= \frac{1}{n^2}\sum_{u=1}^{r}\sum_{v=1}^{r}\sum_{i\in\mathcal{V}_u}\sum_{j\in\mathcal{V}_v} x_i^T y_j \\ &= \frac{1}{n^2}\sum_{u=1}^{r}\sum_{i\in\mathcal{V}_u}\sum_{j\in\mathcal{V}_u} x_i^T y_j + \frac{1}{n^2}\sum_{u=1}^{r}\sum_{v\neq u}\sum_{i\in\mathcal{V}_u}\sum_{j\in\mathcal{V}_v} x_i^T y_j, \\ &= e_x^T\left(\frac{1}{n^2}\sum_{u=1}^{r}\left(n_u G_A^u + n_u(n_u-1)G_B^u\right) + \frac{1}{n^2}\sum_{u=1}^{r}\sum_{v\neq u} n_u n_v\, G_E^{uv}\right) e_y \\ \Leftrightarrow \quad \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i^T y_j &= e_x^T\, G_{C,\mathcal{T}}\, e_y, \end{aligned} \tag{6.43}$$

where matrix $G_{C,\mathcal{T}} \in \mathbb{R}^{p\times p}$ is the sum of all the $p\times p$ blocks of $H_{\mathcal{T}}$ (6.22) divided by n^2:

$$G_{C,\mathcal{T}} = \frac{1}{n^2}\left(\sum_{u=1}^{r} n_u(G_A^u + (n_u-1)G_B^u) + \sum_{u=1}^{r}\sum_{v\neq u} n_u n_v\, G_E^{uv}\right). \tag{6.44}$$

- The average over the n agents of the scalar products of two centered variables related to the same agents:

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^T(y_i-\overline{y})\\
&\quad=\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^Ty_i-x_i^Ty_j)=\frac{1}{n^2}\sum_{u=1}^{r}\sum_{v=1}^{r}\sum_{i\in\mathcal{V}_u}\sum_{j\in\mathcal{V}_v}(x_i^Ty_i-x_i^Ty_j),\\
&\quad=e_x^T\left(\frac{1}{n^2}\sum_{u=1}^{r}n_u(n_u-1)(G_A^u-G_B^u)+\frac{1}{n^2}\sum_{u=1}^{r}\sum_{v\neq u}n_un_v(G_A^u-G_E^{uv})\right)e_y,\\
\Leftrightarrow\quad&\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^T(y_i-\overline{y})=e_x^TG_{D,\mathcal{T}}e_y \qquad (6.45)
\end{aligned}$$

where $\overline{x}=\frac{1}{n}\sum_{i=1}^{n}x_i$, $\overline{y}=\frac{1}{n}\sum_{i=1}^{n}y_i$, and $G_{D,\mathcal{T}}\in\mathbb{R}^{p\times p}$ is given by

$$G_{D,\mathcal{T}}=\frac{1}{n^2}\left(\sum_{u=1}^{r}n_u(n_u-1)(G_A^u-G_B^u)+\sum_{u=1}^{r}\sum_{v\neq u}n_un_v(G_A^u-G_E^{uv})\right) \qquad (6.46)$$

By definitions (6.42), (6.44) and (6.46), we have $G_{A,\mathcal{T}}=G_{C,\mathcal{T}}+G_{D,\mathcal{T}}$. Using relations (6.37), (6.38), (6.40), (6.41), (6.43) and (6.45), we can easily adapt Propositions 6.26-6.33 to this general case with multiple equivalence classes of agents to build all the PEP elements for distributed optimization only based on blocks f_A^u, G_A^u, G_B^u, and G_E^{uv}.

Proposition 6.34 (Function interpolation constraints)**.** *Let $\mathcal{F}_u$ be a set of functions for which there exist linearly Gram-representable interpolation constraints. When considering symmetric agent-class PEP solutions $(\mathbf{f}_{\mathcal{T}}^s-G_{\mathcal{T}}^s)$ (6.19)-(6.20), the set of interpolation constraints for $\mathcal{F}_u$, applied independently to each agent of a given equivalence class $\mathcal{V}_u$, can be expressed with f_A^u and G_A^u.*

Proof. The constraint $f_i\in\mathcal{F}_u$ for each $i\in\mathcal{V}_u$ can be written with a set of Gram-representable interpolation constraints involving function values of agent i and scalar products between quantities (i.e., points and gradients) related to agent i. These are thus single-agent constraints, that are identical for all $i\in\mathcal{V}_u$ when we restrict to symmetrized agent-class solution $(\mathbf{f}_{\mathcal{T}}^s,G_{\mathcal{T}}^s)$ (6.19)-(6.20). Therefore, they can all be expressed with f_A^u and G_A^u using (6.37) and (6.40). □

Algorithm description The class of methods $\mathcal{A}_D$ that we consider is defined in Definition 2.14 and may combine gradient evaluation, consensus

steps, and linear combinations of variables. In the latter, the agents from the same equivalence class should define the same linear combinations of their local variables, otherwise they are not equivalent. Agents from different equivalence classes may apply different combinations. This allows considering heterogeneity in the algorithm parameters, for example, to analyze the effect of uncoordinated step-sizes. In any case, all the variables in the combinations are related to the same agent and can therefore be squared and formulated in PEP using G_A^u, as in (6.40).

Concerning the consensus steps, Theorem 6.22 applies to the set of averaging matrices $\mathcal{W}_{(\lambda^-,\lambda^+)}$, defined in (4.10) in Chapter 4. Necessary interpolation constraints for this set of matrices are given in (4.23), (4.24) and (4.25), from Corollary 4.8. Proposition 6.29 below shows that these interpolation constraints can be expressed in terms of $G_{C,\mathcal{T}}$ (6.44) and $G_{D,\mathcal{T}}$ (6.46), and so in terms of G_A^u, G_B^u, and G_E^{uv} $(u, v = 1, \dots, r)$.

Proposition 6.35 (Averaging matrix interpolation constraints)**.** *The averaging matrix interpolation constraints* (4.23), (4.24) *and* (4.25)*, from Corollary 4.8 can be expressed with G_A^u, G_B^u, and G_E^{uv} $(u, v = 1, \dots, r)$, as:*

$$\begin{aligned}(e_X - e_Y)^T G_{C,\mathcal{T}}(e_X - e_Y) &= 0,\\ (e_Y - \lambda^- e_X)^T G_{D,\mathcal{T}}(e_Y - \lambda^+ e_X) &\preceq 0,\\ e_Y^T G_{D,\mathcal{T}}\, e_X - e_X^T G_{D,\mathcal{T}}\, e_Y &= 0,\end{aligned}$$

where $G_{C,\mathcal{T}}$ and $G_{D,\mathcal{T}}$ are defined in (6.44) *and* (6.46) *and $e_X, e_Y \in \mathbb{R}^p$ are matrices of coefficients such that $P_i e_X = X_i$, $P_i e_Y = Y_i$ with $X_i, Y_i \in \mathbb{R}^{d\times t}$ the matrices with iterates $x_i^k, y_i^k \in \mathbb{R}^d$ as columns.*

Proof. The proof is similar to the one of Proposition 6.29 but using relation (6.43) instead of (6.14) and (6.45) instead of (6.27) □

Points common to all agents As detailed in Appendix 6.A.1 in the case where all agents are equivalent, for any point x_c common to all the agents in the problem, we need two elements in the PEP:

(i) the definition constraints for x_c, e.g., (6.33) or (6.34),

(ii) the interpolation constraints for the new triplet (x_c, g_c, f_c), if g_c or f_c are used in the PEP.

The addition of a new triplet of points to consider in the interpolation constraints does not alter the result of Proposition 6.26 and they can still be

written with f_A^u and G_A^u. The definition constraints can be treated as any other constraint of the problem. When there are several equivalence classes of agents in PEP, this allows more types of constraints to be expressed in the compact PEP formulation, and thus enlarges the type of common points that can be defined in the problem. Optimal point x^* and agent-average $\overline{x}$ can still be expressed in the compact problem, as shown in Propositions 6.36 and 6.38.

Proposition 6.36 (Optimality constraint)**.** *Let x^* be an optimal point for the decentralized optimization problem (2.1). The definition constraints for the common variable x^* in a PEP restricted to symmetrized agent-class solutions $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$ (6.19)-(6.20), given by the system-level stationarity constraint*

$$\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x_i^*) = 0, \quad \text{with } x_i^* = x_j^* = x^* \text{ for all } i, j \in \mathcal{V},$$

can be expressed with G_A^u, G_B^u, and G_E^{uv} $(u, v = 1, \dots, r)$ as

$$e_{g^*}^T \, G_{C,\mathcal{T}} \, e_{g^*} = 0 \quad \text{and} \quad e_{x^*}^T \, G dt \, e_{x^*} = 0,$$

where $G_{C,\mathcal{T}}$ and $G_{D,\mathcal{T}}$ are defined in (6.44) and (6.46) and $e_{g^*}, e_{x^*} \in \mathbb{R}^p$ are vectors of coefficients such that $P_i e_{g^*} = g_i^* = \nabla f_i(x^*)$, and $P_i e_{x^*} = x_i^*$ for all $i = 1, \dots, n$.

Proof. The proof is similar to the one of Proposition 6.30 but using relation (6.43) instead of (6.14) and (6.45) instead of (6.27). □

Remark 6.37. Without loss of generality, the optimal solution x^* of problem (2.1) can be set to $x^* = 0$. In that case, the constraint $x_i^* = x_j^*$, written as $e_{x^*}^T (G_{D,\mathcal{T}}) e_{x^*} = 0$, is not needed to define x^* properly. Indeed, it is sufficient to say that the coefficient vector is zero: $e_{x^*} = 0$. This improves the numerical conditioning of the resulting SDP PEP.

Proposition 6.38 (Agent average definition)**.** *We consider a PEP for distributed optimization, restricted to symmetrized agent-class solutions $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$ (6.19)-(6.20). The definition constraints for $\overline{x}$*

$$\overline{x}_i = \overline{x} = \frac{1}{n}\sum_{j=1}^{n} x_j \quad \text{for all } i \in \mathcal{V},$$

can be written with G_A^u, G_B^u, *and* G_E^{uv} $(u, v = 1, \ldots, r)$ *as*

$$e_{\overline{x}}^T G_{A,\mathcal{T}}\, e_{\overline{x}} + e_x^T G_{C,\mathcal{T}}(e_x - 2e_{\overline{x}}) = 0,$$

where $G_{A,\mathcal{T}}$ *and* $G_{C,\mathcal{T}}$ *are defined in* (6.42) *and* (6.44), *and* $e_{\overline{x}}, e_x \in \mathbb{R}^p$ *are vectors of coefficients such that* $P_i e_{\overline{x}} = \overline{x}_i$, *and* $P_i e_x = x_i$ *for all* $i = 1, \ldots, n$.

Proof. The proof is similar to the one of Proposition 6.32 but using relation (6.41) instead of (6.13) and (6.43) instead of (6.14). □

A point common to all agents, that can only be defined using different equivalence classes of agents, is the iterate of a given agent x_1^t, at which all the local functions are evaluated $f_1(x_1^t), \ldots, f_n(x_1^t)$. This can be used to evaluate the performance of the worst agent, see Section 8.4.1. The definition of such a point is treated in the following proposition.

Proposition 6.39 (Specific agent definition)**.** *We consider a PEP for distributed optimization, restricted to symmetrized agent-class solutions* $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$ (6.19)-(6.20). *Let* $\mathcal{V}_1 = \{1\}$ *be one of the* r *equivalence classes of the PEP. If* x_1 *is a common variable used by all the agents in the PEP, its definition constraints are given by*

$$x_{c,i} = x_1 \quad \text{for all } i \in \mathcal{V} \tag{6.47}$$

and can be written with G_A^u *and* G_E^{uv} $(u, v = 1, \ldots, r)$ *as*

$$e_{x_c}^T G_A^u\, e_{x_c} + e_x^T G_A^1\, e_x - 2e_{x_c}^T G_E^{u1}\, e_x = 0, \qquad \text{for all } u = 2, \ldots, r, \tag{6.48}$$

where $e_{x_c}, e_{x_1} \in \mathbb{R}^p$ *are vectors of coefficients such that* $P_i e_{x_c} = X_{c,i}$, *and* $P_i e_x = x_i$ *for all* $i = 1, \ldots, n$.

Proof. Constraint (6.47) can be written as $(x_{c,i} - x_1)^2 = 0$, which can be expanded as

$$x_{c,i}^T x_{c,i} + x_1^T x_1 - 2x_{c,i}^T x_1 = 0 \quad \text{for all } i \in \mathcal{V}.$$

This constraint can be expressed as (6.48), by using relation (6.40) for the two first terms and definition of G_E^{uv} (6.20) for the third term. □

Initial conditions and performance measures We adapt Proposition 6.33 to the case where there are multiple equivalence classes of agents in the PEP, in order to present usual initial conditions and their reformulation.

The proposition focuses on the initial conditions, but all the expressions involved in these constraints can also be adapted as a performance measure, using the last point x^t instead of the initial point x^0. Other initial conditions or performance measures could involve only one of the equivalence classes of agents.

Proposition 6.40 (Initial conditions)**.** *Let $x^* \in \mathbb{R}^d$ denote the optimal solution of the distributed optimization problem, $x_i^0 \in \mathbb{R}^d$ the initial iterate of agent i, $\overline{x}^0$ the average initial iterate and $R \in \mathbb{R}$ a constant. When the PEP is restricted to symmetric agent-class solutions $(\mathbf{f}_{\mathcal{T}}^s, G_{\mathcal{T}}^s)$ (6.19)-(6.20), here is a list of initial conditions that can be expressed with f_A^u, G_A^u, G_B^u, and G_E^{uv} (for $u, v = 1, \dots, r$):*

Initial condition		*Reformulation*	
$\|x_i^0 - x^*\|^2 \leq R^2,$	$\forall i \in \mathcal{V}$	$(e_{x^0} - e_{x^*})^T G_A^u (e_{x^0} - e_{x^*}) \leq R^2,$	$\forall u$
$\frac{1}{n}\sum_{i=1}^n \|x_i^0 - x^*\|^2 \leq R^2,$		$(e_{x^0} - e_{x^*})^T G_{A,\mathcal{T}} (e_{x^0} - e_{x^*}) \leq R^2,$	
$\|\nabla f_i(x_i^0)\|^2 \leq R^2,$	$\forall i \in \mathcal{V}$	$(e_{g^0})^T G_A^u (e_{g^0}) \leq R^2,$	$\forall u$
$\frac{1}{n}\sum_{i=1}^n \|\nabla f_i(x_i^0)\|^2 \leq R^2,$	$\forall i \in \mathcal{V}$	$(e_{g^0})^T G_{A,\mathcal{T}} (e_{g^0}) \leq R^2,$	
$\frac{1}{n}\sum_{i=1}^n \|x_i^0 - \overline{x}^0\|^2 \leq R^2,$		$(e_{x^0})^T G_{D,\mathcal{T}} (e_{x^0}) \leq R^2,$	
$x_i^0 = x_j^0,$	$\forall i, j \in \mathcal{V}$	$(e_{x^0})^T G_{D,\mathcal{T}} (e_{x^0}) = 0,$	
$\frac{1}{n}\sum_{i=1}^n (f_i(\overline{x}^0) - f_i(x^*)) \leq R,$		$(e_{f(\overline{x}^0)} - e_{f(x^*)})^T f_{A,\mathcal{T}} (e_{f(\overline{x}^0)} - e_{f(x^*)}) \leq R$	

where $f_{A,\mathcal{T}}$, $G_{A,\mathcal{T}}$ and $G_{D,\mathcal{T}}$ are defined in (6.39), (6.42) and (6.46), and $e_{x^0}, e_{x^*}, e_{g^0} \in \mathbb{R}^p$ are vectors of coefficients such that $P_i e_{x^0} = x_i^0$, $P_i e_{x^*} = x^*$ and $P_i e_{g^0} = \nabla f_i(x_i^0)$, for $i = 1, \dots, n$.

Proof. The proof is similar to the one of Proposition 6.33 but using relation (6.40) instead of (6.13), relation (6.45) instead of (6.27) and relation (6.38) instead of (6.12). □

If in doubt, always follow your nose.

Gandalf

7

How to use in practice?

In this chapter, we first describe how to implement our different PEP formulations in practice. For each formulation, code examples have been made available for different algorithms on the GitHub repository Performance-Estimation-Problems-for-distributed-optimization.
In a second time, we explain which formulation to use in which situation, and why. In summary, we recommend using the spectral agent-dependent PEP formulation from Chapter 4 in most cases, because it is easy to implement using the PESTO Matlab toolbox [THG17], we can check its tightness a posteriori, and we can keep the problem small in the many cases where, by Corollary 6.14, we can obtain performance results for any $n \geq 2$ by computing it for $n = 2$.

7.1 Implementation

7.1.1 The agent-dependent PEP formulation

The agent-dependent PEP formulation, from Chapter 4, is the formulation that remains the closest to classical PEP problems. The corresponding SDP can be formulated using the Matlab toolbox PESTO [THG17], or its Python version PEPit [GMG+22], which are dedicated to the formulation of *standard* performance estimation problems. Then, it can be solved with an existing solver, e.g., Mosek.

PESTO and PEPit toolboxes These toolboxes allow writing PEP problems in a very natural manner, i.e., in a high-level description, such as (3.2) and carry out the discretization, the SDP reformulation, and the resolution automatically. In fact, the users only have to provide the performance evaluation settings, but they do not have to know anything about the interpolation conditions or the Gram matrix to be able to formulate and solve the problem. We have contributed to the Matlab PESTO toolbox to make the formulation of the agent-dependent PEPs for decentralized optimization the most user-friendly as possible. Namely, we have embedded in the toolbox the interpolation conditions for the spectral set of averaging matrices $\mathcal{W}_{(\lambda^-,\lambda^+)}$ from Corollary 4.8, so that the users do not need to know them and only specify the set of allowed averaging matrices. Moreover, we have also added auxiliary functions to automatize the repetition of operations over every agent of the problem. Similar contributions still need to be made in the PEPit version.

Code Examples Code examples for the agent-dependent PEP formulation are available in the GitHub repository, for DGD, DIGing, EXTRA, and Acc-DGND, in files `DGD_agents.m`, `DIGing_agents.m`, `EXTRA_agents.m`, and `AccDNGD_agents.m`, respectively.

Problem Size For this agent-dependent formulation, the resulting SDP has a size $\mathcal{O}(tn \times tn)$, where t is the total number of iteration to analyze and n is the number of agents in the problem.

7.1.2 The agent-independent relaxed PEP formulation

The agent-independent relaxed PEP formulation, from Chapter 5, cannot be written easily using the PESTO or PEPit toolbox as it requires ad-hoc adaptations to consider the problem with the new coordinate system and to split the PEP variables according to the two orthogonal components. We have rather implemented it in Matlab using the YALMIP toolbox, which allows building the SDP and calling the Mosek solver to solve it.

Code Examples Code examples for the agent-independent relaxed PEP formulation are available in the GitHub repository, for DGD, DIGing, and EXTRA, in files `DGD_relaxed.m`, `DIGing_relaxed.m`, and `EXTRA_relaxed.m`, respectively.

Problem Size For this agent-independent formulation, the resulting SDP has a size $\mathcal{O}(t \times t)$, where t is the total number of iteration to analyze.

7.1.3 The compact PEP formulation

Similarly, the compact PEP formulation, from Chapter 6, cannot be written easily using the PESTO or PEPit toolbox because the compact SDP works with different variables than standard PEP problems. We have implemented it in Matlab using the YALMIP toolbox, which allows building the SDP and calling the Mosek solver to solve it.

Code Examples Code examples for the compact PEP formulation are available in the GitHub repository, for DGD, DIGing, EXTRA, and Acc-DGND, in files `DGD_symmetrized.m`, `DIGing_symmetrized.m`, `EXTRA_symmetrized.m`, and `AccDNGD_symmetrized.m`, respectively.

Problem Size For this compact PEP formulation, the resulting SDP has a size $\mathcal{O}(tr \times tr)$, where t is the total number of iteration to analyze and r is the number of equivalence classes of agents in the problem.

7.1.4 Examples of running time

All the PEP formulations allows computing the worst-case performance of a decentralized optimization algorithm in a relatively small amount of time. Table 7.1 gives an example of running times of the different PEP formulation for DGD for $n = 2$. As observed in the table, formulations with smaller problem sizes generally translate into shorter computation times.

PEP formulation	SDP size	Running time (in seconds)		
		$t = 5$	$t = 10$	$t = 20$
Exact Agent-dependent (Chapter 4)	$\mathcal{O}(tn \times tn)$	0.87	2.36	8.57
Spectral Agent-dependent (Chapter 4)	$\mathcal{O}(tn \times tn)$	1.29	3.66	17.44
Agent-independent relaxed (Chapter 5)	$\mathcal{O}(t \times t)$	0.44	0.86	2.68
Compact (Chapter 6)	$\mathcal{O}(tr \times tr)$	0.46	0.99	6.59

Table 7.1 Running times to built and solve the different PEP formulation for DGD, in similar settings than Section 4.5, for $n = 2$ agents and $t = 5, 10, 20$ iterations.

7.2 Which formulation to choose?

Let $\mathcal{S}_n$ describe the performance evaluation settings of a decentralized optimization algorithm $A \in \mathcal{A}_D$, see (4.2):

$$\mathcal{S}_n = \{n, A, t, \mathcal{P}, \mathcal{F}, \mathcal{I}, \mathcal{W}\}.$$

In this thesis, we only consider settings that are linearly or LMI Gram-representable (see Definition 4.3). In Chapters 4, 5 and 6, we have developed different PEP formulations to compute (bounds on) $w(\mathcal{S}_n)$, i.e., the worst-case performance of algorithm A in the settings $\mathcal{S}_n$. We refer to the visual outline at the beginning of Part I for a summary of the specificities and relations of the different PEP formulations. This section explains which formulation to use, depending on the settings, to compute the most accurate performance guarantees with the smallest possible problems.

7.2.1 In summary

An important setting for performance evaluation of decentralized optimization is the set of allowed averaging matrices $\mathcal{W}$. We consider that it can either be one given matrix $\mathcal{W} = \{W\}$, or an entire spectral class of averaging matrices $\mathcal{W} = \mathcal{W}_{(\lambda^-, \lambda^+)}$ (4.10). Other sets of matrices could be considered in future works.

A given W To obtain a performance guarantee specific to a given averaging matrix $W \in \mathbb{R}^{n \times n}$, use

the exact agent-dependent formulation (Chapter 4).

Spectral class $\mathcal{W}_{(\lambda^-, \lambda^+)}$ To obtain a performance guarantee valid for any averaging matrix from the spectral class $\mathcal{W}_{(\lambda^-, \lambda^+)}$ (4.10), i.e., any symmetric and generalized doubly stochastic matrix with a given range on the eigenvalues:

(a) If conditions from Theorem 6.13 (part 2) are satisfied, so that we know the worst-case performance $w(\mathcal{S}_n)$ is agent-independent, use

the spectral agent-dependent PEP formulation with $n = 2$ agents. (Chapter 4)

By Corollary 6.14, the obtained worst-case performance guarantee is valid for any number of agents n.

(b) Otherwise, use **the compact PEP formulation** (Chapter 6).

Most of the standard performance evaluation settings in decentralized optimization fall in case (a). Therefore, if you have a doubt, we recommend using the spectral agent-dependent PEP formulation with $n = 2$ agents, which is also the simplest one to implement, especially with the help of the PESTO toolbox.

7.2.2 The detailed reasons

A given W When the averaging matrix (and so the number of agents) is explicitly given, the only formulation that applies is the exact agent-dependent formulation, described in Chapter 4. It provides the exact worst-case performance, specific to the given matrix W.

Spectral class $\mathcal{W}_{(\lambda^-,\lambda^+)}$ When we consider an entire spectral class of averaging matrices $\mathcal{W}_{(\lambda^-,\lambda^+)}$ (4.10), there are three possible PEP formulations. The spectral agent-dependent PEP formulation (Chapter 4), its equivalent compact PEP formulation (Chapter 6) and the agent-independent relaxed PEP formulation (Chapter 5). Table 7.2 summarizes the comparison between these three PEP formulations.

The three PEP formulations are relaxations since they do not exploit the specific Kronecker structure that arises in consensus steps $\mathbf{y}^k = (W \otimes I_d)\mathbf{x}^k$. However, the agent-independent relaxed PEP formulation has another degree of relaxation since it does not exploit the separability of the global objective function in the decentralized problem (2.4), in order to obtain

PEP formulation	SDP size	PESTO	Tightness guarantee
Agent-dependent (Chapter 4)	$\mathcal{O}(tn \times tn)$	✓	check a posteriori
Agent-independent relaxed (Chapter 5)	$\mathcal{O}(t \times t)$	✗	✗
Compact (Chapter 6)	$\mathcal{O}(tr \times tr)$	✗	check a posteriori

Table 7.2 Comparison of our different PEP formulation for analyzing t iterations of a decentralized optimization algorithm with a spectral class of averaging matrices $\mathcal{W}_{(\lambda^-,\lambda^+)}$. As a reminder, n denotes the number of agents and $r \leq n$ the number of equivalence classes of agents in the PEP (see Definition 6.3).

PEP problem with a size independent of the number of agents. Moreover, we know no way of checking the tightness of this formulation a posteriori, even if it has been observed to provide the same (tight) performance results as the others in our experiments. Therefore, we do not recommend using the agent-independent relaxed PEP formulation from Chapter 5 in practice, even if its size is independent of the number of agents, and so often smaller than the size of the agent-dependent PEP formulation.

The spectral agent-dependent PEP formulation (Chapter 4) and the compact PEP formulation (Chapter 6) give equivalent results, by construction. Both formulations allow checking the tightness of their results a posteriori, by recovering an instance of worst-case averaging matrix in $\mathcal{W}_{(\lambda^-,\lambda^+)}$. The size of the SDP for the compact PEP formulation only depends on the number of equivalence classes of agents in the problem and not directly on the total number of agents, as the agent-dependent formulation. Therefore, the compact PEP formulation is often much smaller than the agent-dependent PEP. However, when the performance evaluation settings $\mathcal{S}_n$ satisfy conditions from Theorem 6.13 (part 2), so that we know the worst-case performance $w(\mathcal{S}_n)$ is agent-independent, we recommend using the spectral agent-dependent PEP formulation with $n = 2$ agents. When $n = 2$, the agent-dependent PEP problem is as small as the compact PEP formulation, and by Corollary 6.14, the obtained worst-case performance guarantee is valid for any number of agents $n \geq 2$, including $n \to \infty$. Moreover, such an agent-dependent PEP is more intuitive to formulate and implement, especially with the user-friendly PESTO toolbox. Finally, the current solvers, such as Mosek, will perform better on the agent-dependent PEP formulation, than on the compact one.

The Mosek solver, like most of the other existing ones, relies on the interior-point method and the existence of a Slater point in the feasible set of the problem. Some equality constraints in the compact SDP PEP problems may prevent the existence of a Slater point, hence leading the solver to find solutions of low numerical quality. Indeed, the compact PEP formulation, described in Chapter 6, and detailed in Appendix 6.A, contains equality constraints that impose rank deflection of the matrix variables of the resulting SDP, see for example, the optimality constraints, or the average preserving condition for consensus steps (Propositions 6.30 and 6.29 in Appendix 6.A). This may be problematic for the numerical conditioning of the problem, and thus the numerical quality of its solution. We have verified the accuracy of the worst-case value given by the compact PEP formulation for $n \leq 5$ for different algorithms and settings (DGD, DIGing,

and EXTRA), by comparing its results with those of the agent-dependent PEP formulation. In all our experiments, we have observed a relative error of less than 1%. However, the worst-case solutions (i.e., functions and iterates) can be very high-dimensional.

In the agent-dependent PEP formulations, all the problematic equality constraints can easily be removed, by exploiting them to reduce the size of the SDP. This allows the problem and its solution to be well-conditioned. Such dimensionality reductions are more involved in the compact PEP SDP formulations. We hope that existing toolboxes and solvers will be extended to automatically reduce the SDP size by exploiting equality constraints. Therefore, for now, if one wishes to evaluate the performance of a distributed optimization algorithm in settings for which we know the result will be independent of n by Theorem 6.13, we recommend solving the agent-dependent formulation with $n = 2$ because the current solvers will perform better on this version of the problem.

PART II
Exploitation for analysis and tuning

Stude, non ut plus aliquid scias, sed ut melius.
Study, not to know more, but to know better.

Seneca

8

Algorithms analysis

USING THE PEP FORMULATIONS from Part I, we can analyze the worst-case performance of any decentralized optimization algorithm in $\mathcal{A}_D$, in various settings. As explained in Chapter 7, we will only rely on the agent-dependent and the compact PEP formulations, so that we can check their tightness a posteriori. These formulations enable rapid and automatic computation of numerical performance bounds, whose tightness has been checked a posteriori in our experiments.

There are many decentralized optimization algorithms tailored to solve Problem 2.1, see Section 2.4 for a review. In this chapter, we propose to analyze the performance of a few algorithms that are significant in the literature:

- **DGD** [NO09] with non-smooth and Lipschitz local functions in Section 8.1. Our PEP analysis improves on theoretical guarantees, including for the step-size tuning, and recovers the worst-case averaging matrix.
- **DIGing** [NOS16] with smooth and strongly convex local functions in Section 8.2. We show that its worst-case performance is independent of the number of agents n. Moreover, we analyze its convergence rate with PEP and propose a tuning of the step-size which largely improves the performance of the algorithm. The resulting worst-case averaging matrices are the same as for DGD.
- **Acc-DGND** [QL19] with smooth and convex local functions in Sec-

tion 8.3. We use our PEP tool to show that the $\mathcal{O}(1/t^2)$ convergence conjecture of [QL19] seems to hold if the step-size is correctly chosen.

- **EXTRA** [SLWY14] with smooth and strongly convex local functions in Section 8.4. We leverage the compact PEP formulation to analyze the algorithm in relevant settings that have never been studied in the literature, such as the performance of the worst agent, the performance of the 80-th percentile of the agent performance, and the average performance under heterogeneity in the local functions conditioning. Unlike many settings, these worst-case performances do depend on the number of agents n. We analyze their evolution with n, including when n goes to infinity.

Finally, Section 8.5 compares these algorithms, and a few others, in equivalent and fair settings. In particular, we show that ATC-DIGing [NOSU17] outperforms DIGing, EXTRA is not robust to time-varying averaging matrices, unlike the other analyzed algorithms, and that the accelerated algorithms Acc-DNGD and its recent variant Acc-GT [LL21] may have large hidden constant, as they do not perform better than EXTRA or ATC-DIGing after $t = 10$, $t = 20$ and $t = 30$ iterations.

This chapter is partially based on sections of [CH23] and [CH24a].

8.1 Distributed (sub)gradient descent (DGD)

We start by analyzing one of the first and simplest decentralized optimization methods, namely, the distributed gradient descent DGD [NO09]. While providing similar asymptotic rates, our PEP analysis outperforms the theoretical performance bounds, allows for better tuning of the step-size, and recovers the worst-case averaging matrix.

We consider t iterations of DGD, described in Algorithm 1, applied on n agents and with a constant step-size α and a constant averaging matrix W. As explained in Section 2.4, there are different studies on DGD; e.g., [NO09, SRNV12, YLY13]. In [YLY13], the authors showed that its iterates converge to a neighborhood of the optimal solution x^* when the step-size is constant. In the sequel, we take as baseline the results of a recent survey [NOR18] providing a theoretical bound for the functional error of the running average iterate, valid when the subgradients are bounded.

Theorem 8.1 (Performance bound for DGD [NOR18]). *Let $f_i, \dots, f_n \in \mathcal{F}_R$, i.e., convex functions with subgradients bounded by R (see Definition 3.3). Let x^0 be an identical starting point for all agents such that $\|x^0 - x^*\|^2 \leq D^2$. And let $W \in \mathcal{W}_{(-\lambda,\lambda)}$ for some $\lambda \in [0,1)$, i.e., W is a symmetric and generalized doubly stochastic matrix with eigenvalues $\lambda_2, \dots, \lambda_n \in [-\lambda, \lambda]$, see (4.10).*
If we run DGD for t steps with a constant step-size $\alpha = \frac{1}{\sqrt{t}}$ and averaging matrix W, then there holds[1]

$$f(x_{\text{av}}) - f(x^*) \leq \frac{D^2 + R^2}{2\sqrt{t}} + \frac{2R^2}{\sqrt{t}(1-\lambda)}, \tag{8.1}$$

where $x_{\text{av}} = \frac{1}{n(t+1)} \sum_{i=1}^{n} \sum_{k=0}^{t} x_i^k$ is the average over all the iterations and all the agents.

Theorem 8.1 was stated in [NOR18, Theorem 8] for doubly stochastic matrix W but the proof never uses the non-negativity assumption and therefore it also holds for generalized doubly stochastic matrices, that we consider in this thesis. Moreover, the theoretical guarantee remains valid if the averaging matrix is time-varying. We use a time-constant averaging matrix here because it makes it easier to check the tightness of the PEP bounds a posteriori, see Section 4.3.3, and it does almost not change the resulting PEP bound for DGD.

For comparison purposes, we will analyze the performance of DGD using our PEP formulations in the same settings as Theorem 8.1. These settings are all Gram-representable and satisfy conditions of Theorem 6.13 (part 2), i.e., performance criterion and initial conditions are scale-invariant (Definition 6.9). Therefore, we know that the worst-case performance in these settings is independent of the number of agents when $n \geq 2$, so we can focus our experiments on the basic case $n = 2$, without loss of generality. This also confirms observations on agent independence from previous DGD numerical experiments in Section 4.5. Hence, as explained in Chapter 7, our PEP analysis will rely on the spectral agent-dependent PEP formulation with $n = 2$. We refer to Chapter 4 for details about this formulation. In particular, Section 4.4 presents an example for DGD.

In our experiments, we also focus on the situation where $D = 1$ and $R = 1$, but the results obtained can be scaled up to general values using changes of variables, as explained in Section 8.1.5.

[1] Note the factor 2 in the second term of the bound (8.1) was missing in [NOR18], but its presence was confirmed by the authors of [NOR18].

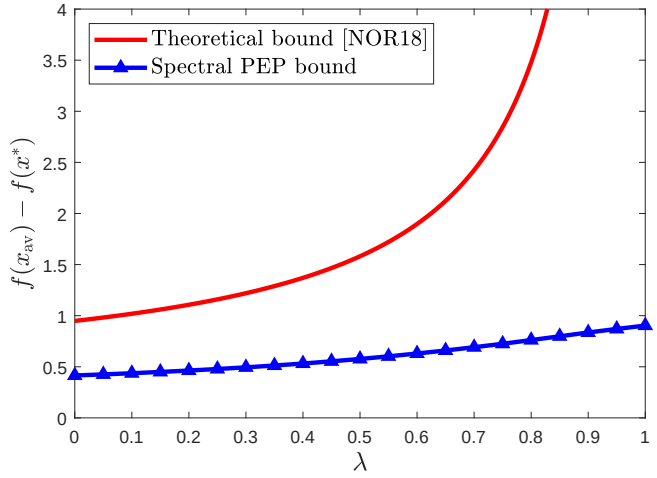


Fig. 8.1 Evolution with λ of the worst-case performance of $t = 10$ iterations of DGD in the setting of Theorem 8.1 with $n = 2$ agents. The plot shows the theoretical bound from equation (8.1) (in red), largely above the spectral PEP worst-case performance (in blue).

8.1.1 Comparison with Theorem 8.1

We compare the spectral PEP bound with the theoretical bound from Theorem 8.1 for different ranges of eigenvalues $[-\lambda, \lambda]$ for the averaging matrix. The value of $1 - \lambda$ corresponds to the algebraic connectivity of the network, as detailed in Section 2.3.2. Fig. 8.1 shows the evolution of both bounds with λ for $t = 10$ iterations of DGD with $n = 2$ agents. We observe that the spectral worst-case performance bound (in blue) largely improves on the theoretical one (in red), especially when λ approaches 1, in which case the theoretical bound grows unbounded. Improvements to the bounds when λ is close to 1 are particularly relevant since large values for λ are frequent for averaging matrices of large networks of agents [BBP19]. For example, for a 5 by 5 grid of agents with Metropolis weights [BDX04, BGPS06, NOR18], the range of eigenvalues of the resulting averaging matrix is $[-0.92, 0.92]$. In that case, after $t = 10$ iterations, our spectral bound guarantees that the performance measure is below 0.85, compared to 8.2 for the theoretical bound from Theorem 8.1. This accuracy of 0.85 would only be guaranteed using Theorem 8.1 with $t = 936$.

8.1.2 Worst-case averaging matrix and tightness analysis

As detailed in Chapter 4, we can check tightness of the spectral PEP formulation a posteriori by recovering a valid worst-case averaging matrix $W \in \mathcal{W}_{(-\lambda,\lambda)}$. This has been done for DGD in such settings in Section 4.5.2. The identified valid worst-case averaging matrix always have this form

$$\hat{W}^* = (1+\lambda)\frac{\mathbf{1}\mathbf{1}^\mathbf{T}}{n} - \lambda I. \tag{8.2}$$

As detailed in Section 4.5.2, the matrix $\hat{W}^*$ is well in $\mathcal{W}_{(-\lambda,\lambda)}$ and has the following set of eigenvalues: $\{1, -\lambda, \dots, -\lambda\}$. Therefore, for the same performance evaluation settings as Theorem 8.1, the spectral PEP bounds for DGD, are tight, meaning that there is no better (i.e., smaller) bound on the worst-case performance of DGD in these settings.

8.1.3 Evolution with the total number of iterations t

Fig. 8.2 shows the evolution of the spectral PEP worst-case performance for DGD multiplied by $\sqrt{t}$, for different values of λ. Except when $\lambda = 1$, all lines tend to a constant value, meaning that the spectral bound behaves in $\mathcal{O}\left(\frac{1}{\sqrt{t}}\right)$, as the theoretical bound (8.1), but with a much smaller multiplicative constant. When $\lambda = 1$, the line grows linearly and never reaches a constant value. In that case, the worst-case averaging matrices lead to counterproductive interactions, preventing DGD from working in the worst case.

Convergence in $\mathcal{O}\left(\frac{1}{\sqrt{t}}\right)$ for DGD with convex (possibly) non-smooth functions with bounded subgradients, is as good as the standard convergence result for the subgradient method, in a centralized setting [Nes18].

8.1.4 Tuning the step-size α

The PEP methodology allows us to easily tune the parameters of a method. For example, Fig. 8.3 shows the evolution of the spectral PEP worst-case performance of DGD with the constant step-size it uses, in the setting of Theorem 8.1 with $n = 2$, $t = 10$ and $\lambda = 0.8$. In that case, we observe that the value $\alpha = \frac{1}{\sqrt{t}}$ used in Theorem 8.1 for deriving the theoretical performance bound is not the best possible choice for the step-size and should be divided by two to improve the performance guarantees by 30%. The optimal value for the step-size regarding our spectral bound, denoted

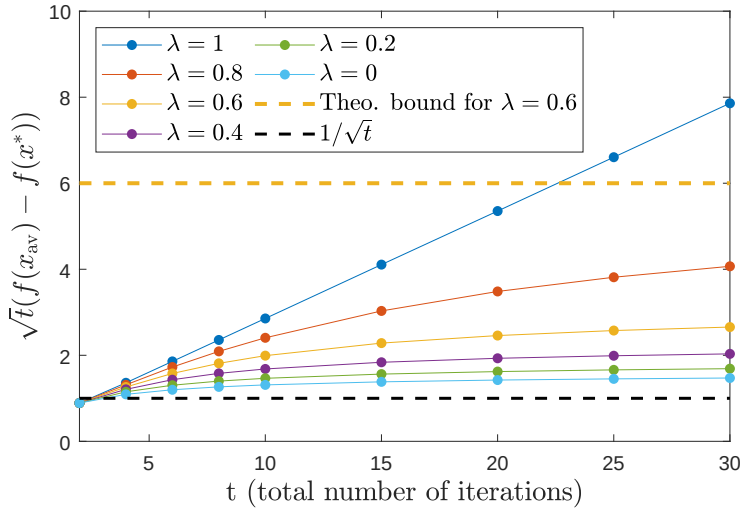


Fig. 8.2 Evolution with t of the *normalized* spectral PEP worst-case performance of t iterations of DGD in the setting of Theorem 8.1 with $n = 2$ and different value of λ. Theoretical bound (8.1) is also shown for $\lambda = 0.6$. The shown bounds are normalized by $\frac{1}{\sqrt{t}}$ to show that they evolve at this rate.

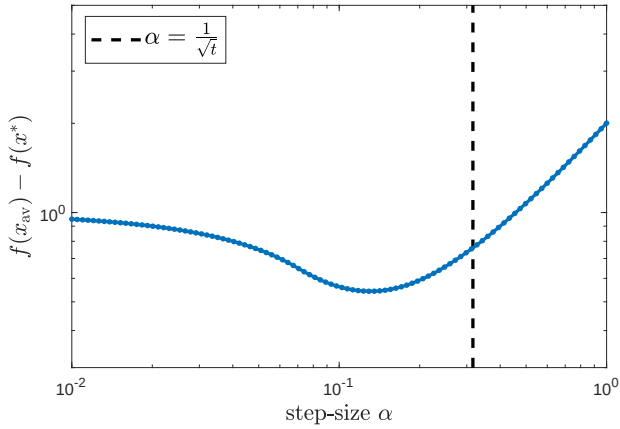


Fig. 8.3 Evolution with α of the spectral worst-case performance of $t = 10$ iterations of DGD in the setting of Theorem 8.1 with $n = 2$ agents and $\lambda = 0.8$ (except for α).

α^*, provides the best worst-case guarantee, whatever the averaging matrix from $\mathcal{W}_{(-\lambda,\lambda)}$ is used. The value of α^* depends on λ and t. Further numerical explorations are needed to guess an analytical expression for α^*, but we have observed that it decreases when either t or λ increases.

The impact of the step-size on the other experiments and observations can be studied by setting $\alpha = \frac{h}{\sqrt{t}}$, for some $h > 0$. We focused on $h = 1$ for comparison with the theoretical bound from Theorem 8.1. Nevertheless, all our other observations have been confirmed[2] for $h = 0.1, 0.5, 2, 10$.

8.1.5 Note on scaling of DGD

In the DGD analysis, we have one constant R to bound the subgradients $\|g_i^k\|^2 \leq R^2$ (for $k = 0, \ldots, t$), and another one D to bound the initial distance to the optimum $\|x^0 - x^*\|^2 \leq D^2$. In our performance estimation problem, we consider general positive values for these parameters $D > 0$ and $R > 0$ and we parametrize the step-size by $\alpha = \frac{Dh}{R\sqrt{t}}$, for some $h > 0$. To pass from this general problem to the specific case where $D = 1$ and $R = 1$, that we actually solve, we consider the following changes of variables:

$$\tilde{x}_i = \frac{x_i}{D}, \quad \tilde{f}_i(\tilde{x}_i) = \frac{1}{DR} f_i(x_i) \quad \text{and} \quad \tilde{\alpha} = \frac{R\alpha}{D}.$$

These changes of variables do not alter the updates of the algorithm or the nature of the problem. They allow expressing the worst-case guarantee obtained for $f(x_{\text{av}}) - f(x^*)$ with general values of D, R, and h, denoted $w(D, R, h)$, in terms of the worst-case guarantee obtained for $\tilde{f}(\tilde{x}_{\text{av}}) - \tilde{f}(\tilde{x}^*)$ with $D = R = 1$, denoted $\tilde{w}(1, 1, h)$:

$$w(D, R, h) = DR\, \tilde{w}(1, 1, h). \tag{8.3}$$

The same kind of scaling can be applied to Theorem 8.1. The theorem is valid for general values of D and R but is specific to $\alpha = \frac{1}{\sqrt{t}}$, which is equivalent to picking $h = \frac{R}{D}$. After the scaling, we obtain the following bound, valid for $D = 1$, $R = 1$, and any value of $\alpha = \frac{h}{\sqrt{t}}$ with $h > 0$:

$$\tilde{f}(\tilde{x}_{\text{av}}) - \tilde{f}(\tilde{x}^*) \leq \frac{h^{-1} + h}{2\sqrt{t}} + \frac{2h}{\sqrt{t}(1 - \lambda)}. \tag{8.4}$$

[2] For too small step-size such as $h < 0.1$, the worst-case averaging matrix is no more given by (8.2).

This scaled theoretical bound is shown as a function of $\alpha = \frac{h}{\sqrt{t}}$, in comparison with the PEP bound, in Fig. 1.2b in the introduction. When $h = 1$, the bound is equivalent to the one from Theorem 8.1 with $D = R = 1$, which was the focus of the numerical analysis in this section.
The bound (8.4) can be extended to any value of $D > 0$ and $R > 0$, using the relation from equation (8.3):

$$f(x_{\text{av}}) - f(x^*) \leq DR\left(\frac{h^{-1} + h}{2\sqrt{t}} + \frac{2h}{\sqrt{t}(1-\lambda)}\right).$$

8.2 DIGing

The DIGing algorithm is one of the first proposed algorithms that converges linearly with a constant step-size, for smooth and strongly convex functions. Moreover, it also introduces the idea of *gradient tracking* which has become a frequent technique in decentralized optimization algorithms. Our PEP analysis reveals that the worst-case performance of DIGing is independent of the number of agents n, unlike its theoretical guarantees from [NOS16]. Moreover, it allows tuning the step-size correctly, and uniformly for all system sizes, and it largely improves on the existing theoretical guarantees. We also recover the worst-case averaging matrices and confirm the robustness of the algorithm to time-varying networks.

The DIGing algorithm [NOS16], described in Algorithm 3, combines DGD with a *gradient tracking* technique. Each agent i holds an estimation s_i of the average of all the local gradients and uses it in its update instead of its own local gradient. The DIGing algorithm allows using a different averaging matrix W^k at each iteration k.

The linear convergence of DIGing has been established in [NOS16, Theorem 3.14] provided that the local functions f_i are L-smooth and μ-strongly convex (with $L \geq \mu > 0$); the averaging matrices are symmetric, doubly stochastic, and have their second-largest eigenvalue λ below 1 (in absolute value); and the step-size is within the interval

$$\alpha \in \left(0, \frac{(1-\lambda)^2}{2L\left(1 + 4\sqrt{n}\sqrt{L/\mu}\right)}\right]. \tag{8.5}$$

The largest accepted step-size decreases thus as $\mathcal{O}(\frac{1}{\sqrt{n}})$ and has also a dependence in $\mathcal{O}(\frac{1}{L\sqrt{L/\mu}})$ which is less favorable than the usual $\mathcal{O}(\frac{1}{L})$ in opti-

mization, leading thus often to very small values for accepted α. The spectral condition on the averaging matrices ($|\lambda| < 1$) guarantees the network connectivity at each iteration (see Proposition 2.12). Actually, [NOS16] imposes a weaker spectral condition which only requires B-connected sequence of networks (see Definition 2.13). In this section, we consider the case $B = 1$. Under all these conditions, [NOS16, Theorem 3.14] guarantees the following R-linear convergence:

$$\sqrt{\sum_{i=1}^{n} \|x_i^t - x^*\|^2} \leq C\rho_{\text{theo}}^t, \quad \text{for all } t \geq 0, \tag{8.6}$$

where C is a positive constant and $\rho_{\text{theo}} \in (0,1)$ is the convergence rate depending on n, λ, L and μ (see [NOS16, Theorem 3.14] for details about its expression).

R-linear and Q-linear convergence

We recall the definition of the R-linear convergence and the differences with the Q-linear convergence, based on the definitions provided in [NOS16]. Suppose that a sequence $\{x^k\}$ converges to x^* in some norm $\|\cdot\|$. We say that the convergence is:

(i) R-linear if there exists $\rho \in (0,1)$ and some positive constant C such that $\|x^k - x^*\| \leq C\rho^k$ for all k;

(ii) Q-linear if there exists $\rho \in (0,1)$ such that $\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|} \leq \rho$ for all k.

Both convergences are geometric but the Q-linear convergence is stronger since it implies a monotonic decrease of $\|x^k - x^*\|$, while R-linear convergence does not. By definition, Q-linear convergence implies R-linear convergence with the same rate but the inverse implication does not hold in general.

This section analyzes the worst-case performance of DIGing using our new PEP formulations, in the same settings as [NOS16, Theorem 3.14] to get a fair comparison. Thus, we consider the set $\mathcal{F}_{\mu,L}$ of L-smooth and μ-strongly convex functions for local functions (Definition 3.2). As performance criterion, we consider the same as in (8.6) but squared and scaled

by n, so that it is scale-invariant (see Definition 6.9):

$$\mathcal{P}^t = \frac{1}{n}\sum_{i=1}^{n} \|x_i^t - x^*\|^2. \tag{8.7}$$

The corresponding theoretical convergence rate for this criterion is given by ρ^2_{theo}. In DIGing (Algorithm 3), the average gradient estimates are initialized as

$$s_i^0 = \nabla f_i(x_i^0), \qquad \text{for } i = 1, \dots, n.$$

We also consider initial conditions similar to those used implicitly in [NOS16, Theorem 3.14]:

$$\frac{1}{n}\sum_{i=1}^{n} \|x_i^0 - x^*\|^2 \leq D^2, \tag{8.8}$$

$$\frac{1}{n}\sum_{i=1}^{n} \|s_i^0 - \frac{1}{n}\sum_{j=1}^{n} \nabla f_j(x_j^0)\|^2 \leq E^2. \tag{8.9}$$

Condition (8.8) bounds the initial performance criterion $\mathcal{P}(x^0)$ which measures the average error of the initial iterates x_i^0. Condition (8.9) bounds the average error made by the agents on the initial average gradient estimates s_i^0. The performance criterion and the two initial conditions are scale-invariant, so that, by Theorem 6.13, we know the resulting worst-case performance is independent of the number of agents $n \geq 2$. Hence, as explained in Chapter 7, our PEP analysis will rely on the spectral agent-dependent PEP formulation with $n = 2$.

To compare our results to the theoretical guarantee from [NOS16, Theorem 3.14], we consider time-varying averaging matrices $W^k \in \mathcal{W}_{(-\lambda,\lambda)}$ for $\lambda \in [0, 1)$, i.e., each matrix is symmetric, generalized doubly stochastic, and has all its eigenvalues in $[-\lambda, \lambda]$, except $\lambda_1 = 1$, see (4.10). In a second time, we will also consider constant averaging matrices in $\mathcal{W}_{(-\lambda,\lambda)}$. We refer to Chapter 4 for details about the spectral agent-dependent PEP formulation. In particular, Section 4.4 presents an example for DGD, and explains how to impose a time-constant averaging matrix or how to let it vary over time in the PEP.

The PEP problem depends on 6 parameters: L, μ, D, E, λ, α. We fix $L = 1$ and $D = 1$, as the results can then be scaled up to general values using appropriate changes of variables, as explained for DGD in Section 8.1.5. The value of E is arbitrarily fixed to $E = 1$, but all the observations have been confirmed for other values of E ($E = 0.1$ and $E = 10$). Different values of the step-size α will be analyzed to understand its impact on the

worst-case performance of the DIGing algorithm. We show the results for representative values of μ and λ ($\mu = 0.1$ and $\lambda = 0.9$).

8.2.1 Impact of the number of agents n

As observed in Section 4.5 (Fig. 4.2), and predicted by Theorem 6.13, the spectral PEP worst-case of DIGing in the described settings is independent of the number of agents n (for $n \geq 2$). This differs from the theoretical analysis of DIGing from [NOS16, Theorem 3.14] for which the range of accepted step-sizes, as well as the convergence rate, depend on n. It appears that the worst-case performance of DIGing does not get worse when n increases, and can be bounded uniformly for all values of n, for example with our spectral PEP bound. Such uniform bound will allow us to better choose the step-size α of DIGing, identically for all n. For the subsequent analysis of the results of our spectral PEP formulation for DIGing, we fix $n = 2$ for keeping a low computational complexity. Corollary 6.14 ensures that the worst-case performance obtained with $n = 2$ is also valid for all $n \geq 2$. The case $n = 2$ corresponds to the most favorable situation for the theoretical results [NOS16, Theorem 3.14], which get worse as n increases.

8.2.2 Comparison between the theoretical and the spectral PEP bounds

We compare the spectral bounds obtained with PEP with the corresponding guarantees obtained using ρ^2_{theo}, i.e., the square of the theoretical convergence rate bound [NOS16, Theorem 3.14]. Fig. 8.4 shows the evolution of both guarantees (spectral and theoretical) with the total number of iterations t of the algorithm, for different values of the step-size α and for $n = 2$, $\lambda = 0.9$ and $\mu = 0.1$. The spectral bounds are always smaller than the corresponding theoretical ones.

For the three values of α, we observe a linear decrease of the spectral worst-cases, which strongly suggests a linear convergence rate. The observed rates are listed in TABLE 8.1 below and can be compared with the theoretical convergence rate ρ^2_{theo}, which are all larger. The step-size $\alpha = 2.6 \times 10^{-4}$ is the one that optimizes the theoretical convergence guarantee ρ_{theo} from [NOS16, Theorem 3.14] for $n = 2$. For some step-sizes, such as $\alpha = 10^{-3}$, convergence is not guaranteed by [NOS16, Theorem 3.14], even when $n = 2$, while our observations suggest that it does occur. And this gets worse when n becomes larger since it would require the the-

oretical step-sizes to be smaller, i.e., $\alpha_{\max} \approx \frac{4\times 10^{-4}}{\sqrt{n}}$ with $L = 1$, $\mu = 0.1$ and $\lambda = 0.9$, see (8.5). Therefore, the spectral bound for DIGing helps to significantly improve the choice of step size.

The same observations can be done for other values of μ and λ. In particular, other values of μ lead to the same graphs, with a different scale for the vertical axis. We test it for $\mu = 0.01$ and $\mu = 0.001$.

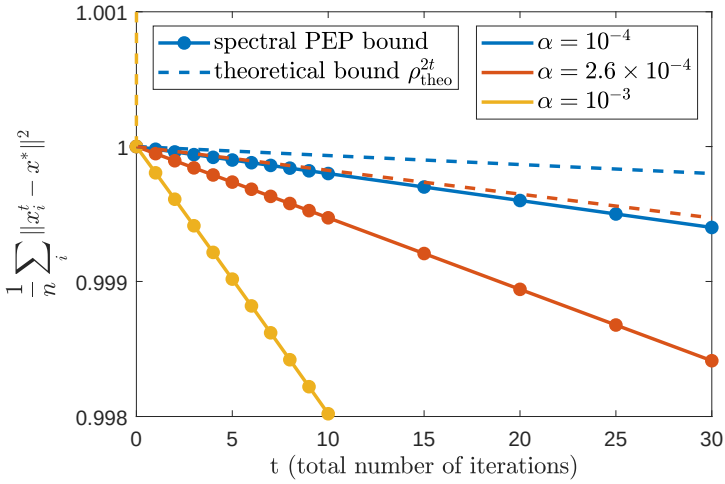


Fig. 8.4 Evolution with t of the spectral worst-case of t iterations of DIGing with $n = 2$, $\lambda = 0.9$, $\mu = 0.1$, and different values of step-size α. The corresponding theoretical rates from [NOS16, Theorem 3.14] are also shown in comparison. Logarithmic y-axis.

step-size α	observed rate with PEP	theoretical rate ρ_{theo}^2
10^{-4}	$1 - 2 \times 10^{-5}$	$1 - 7 \times 10^{-6}$
2.6×10^{-4}	$1 - 5 \times 10^{-5}$	$1 - 2 \times 10^{-5}$
10^{-3}	$1 - 2 \times 10^{-4}$	/

Table 8.1 Theoretical [NOS16] and observed spectral PEP rates for $n = 2$, $\lambda = 0.9$, $\mu = 0.1$ and for different step-sizes α.

8.2.3 Convergence rate analysis with PEP

Fig. 8.4 strongly suggests a linear convergence rate but the performance guarantees only hold for the values of t tested and we cannot extrapolate them with certainty, e.g., the performance could explode after a larger number of iterations. We now show how to use our PEP formulations to obtain a *guaranteed convergence rate* valid for any number of iterations. The idea is to use the same metric as an initial condition and performance measure to be able to consider the problem over only one general DIGing iteration. We also need to ensure that the DIGing update preserved the assumptions on initial conditions. The metric we use is the weighted combination of the two error measures (8.8) and (8.9) previously used separately in the initialization:

$$\mathcal{P}_\gamma^t = \frac{1}{n}\sum_{i=1}^n \|x_i^t - x^*\|^2 + \frac{\gamma}{n}\sum_{i=1}^n \|s_i^t - \frac{1}{n}\sum_{j=1}^n \nabla f_j(x_j^t)\|^2, \tag{8.10}$$

where γ is a positive weighting coefficient.

Proposition 8.2 (Convergence rate of DIGing with PEP)**.** *Consider a spectral agent-dependent PEP formulation of one iteration of DIGing with $\mathcal{P}_\gamma^1$ as performance criterion and with the following initialization: pick any $x_i^0, s_i^0 \in \mathbb{R}^d$ such that*

$$\sum_{i=1}^n s_i^0 = \sum_{i=1}^n \nabla f_i(x_i^0) \qquad \text{and} \qquad \mathcal{P}_\gamma^0 = 1.$$

Let θ_γ be the optimal value of this PEP, then

$$\mathcal{P}^k \leq \mathcal{P}_\gamma^k \leq \theta_\gamma^k \mathcal{P}_\gamma^0 \qquad \text{for all } \gamma \geq 0 \text{ and } k = 0, 1, \ldots$$

Convergence is Q-linear[3] *for $\mathcal{P}_\gamma^k$ (8.10) and R-linear for $\mathcal{P}^k$ (8.7), both with convergence rate θ_γ depending on coefficient γ.*

Proof. Let M be a non-negative coefficient. One can verify that the following changes of variables,

$$\tilde{x}_i = \sqrt{M} x_i, \quad \tilde{s}_i = \sqrt{M} s_i \quad \text{and} \quad \tilde{f}_i(\tilde{x}_i) = M f_i(x_i),$$

do not affect the behavior of DIGing and scale both $\mathcal{P}_\gamma^0$ and $\mathcal{P}_\gamma^1$ by a factor M:

$$\tilde{\mathcal{P}}_\gamma^0 = M\mathcal{P}_\gamma^0, \qquad \tilde{\mathcal{P}}_\gamma^1 = M\mathcal{P}_\gamma^1. \tag{8.11}$$

[3] See definition of Q-linear and R-linear convergence in the above remark box

Since $\mathcal{P}_\gamma^0 = 1$, this means that $M = \tilde{\mathcal{P}}_\gamma^0$. Therefore, by definition of θ_γ as the optimal (maximal) value of $\mathcal{P}_\gamma^1$, equation (8.11) leads to

$$\tilde{\mathcal{P}}_\gamma^1 \leq \theta_\gamma \tilde{\mathcal{P}}_\gamma^0. \tag{8.12}$$

Equation (8.12) holds for any value of $\tilde{\mathcal{P}}_\gamma^0 \geq 0$ (e.g., for $\mathcal{P}_\gamma^k$). Moreover, the DIGing updates are the same for all iterations, and thus, inequality (8.12) is valid for any iteration k

$$\mathcal{P}_\gamma^{k+1} \leq \theta_\gamma \mathcal{P}_\gamma^k,$$

provided that iterates x_i^k, s_i^k also satisfy the initial condition

$$\sum_{i=1}^n s_i^k = \sum_{i=1}^n \nabla f_i(x_i^k), \qquad \text{for any } k. \tag{8.13}$$

This condition (8.13) holds by assumption for $k = 0$ and is preserved by a DIGing update with a valid averaging matrix W (see Algorithm 3 and Definition 2.7), as

$$\begin{aligned}
\sum_{i=1}^n s_i^{k+1} &= \sum_{i=1}^n \sum_{j=1}^n w_{ij}^k s_j^k + \sum_{i=1}^n \nabla f_i(x_i^{k+1}) - \sum_{i=1}^n \nabla f_i(x_i^k) \\
&= \sum_{j=1}^n s_j^k + \sum_{i=1}^n \nabla f_i(x_i^{k+1}) - \sum_{i=1}^n \nabla f_i(x_i^k) \\
&= \sum_{i=1}^n \nabla f_i(x_i^{k+1}).
\end{aligned}$$

Finally, by definition of $\mathcal{P}^k$ (see (8.7)) and $\mathcal{P}_\gamma^k$, we have well that $\mathcal{P}^k \leq \mathcal{P}_\gamma^k$ for any $k, \gamma \geq 0$. □

Using Proposition 8.2, we can obtain guaranteed convergence rates for DIGing, which depend on the weighting coefficient γ, for the metric $\mathcal{P}_\gamma^t$. These convergence rates are also valid for $\mathcal{P}^t$, for all $\gamma \geq 0$, and can thus be compared with the observed rates from Fig. 8.4 and the theoretical convergence rates (8.6) ([NOS16, Theorem 3.14]). An exploration of the different values for the weighting coefficient $\gamma \geq 0$ suggests that the best rates are obtained for $\gamma = \frac{\alpha}{L}$ but other rates are also valid. With $\gamma = \frac{\alpha}{L}$, we recover exactly the same rates as those observed in Fig. 8.4, but they are guaranteed with certainty for any number of iterations t. The spectral agent-dependent PEP problem from Proposition 8.2 has a small size since it only

considers one iteration, and is thus rapidly solved. The size of the problem still increases with the number of agents n but, in many settings, including those analyzed in this section, solving it for $n = 2$ provide a worst-case performance valid for all $n \geq 2$, see Corollary 2.10.

The above approach can be applied to other algorithms provided that their updates are identical for all iterations. It presents some parallels with the Lyapunov approach used in [SVSL20]. Both approaches analyze the decrease of a particular function over only one iteration. We design the decreasing criterion $\mathcal{P}_\gamma$ by hand and have optimized the value of γ to find the smallest rate. The Lyapunov approach [SVSL20] allows easily optimizing the rate over a wider class of Lyapunov functions, which may result in smaller rates. On the other hand, our PEP approach provides the worst-case functions and averaging matrices resulting from the worst-case solutions. It also allows comparing what happens over one iteration and over several, and with averaging matrices that are variable or constant in time.

8.2.4 Impact of the step-size α

Fig. 8.5 compares the spectral rates, obtained with the spectral PEP formulation from Proposition 8.2 with the theoretical ones from [NOS16, Theorem 3.14] for a wide range of values for the step-size α. The spectral PEP rates are identical for any value of n and present a first regime where they decrease as α increases until a certain threshold step-size α^*. This decrease is numerically matching $1 - 2\mu\alpha$ for all values of λ we tested. After α^*, we observe a sharp increase in the spectral PEP rates. For $\mu = 0.1$, this threshold step-size has been identified numerically to be given by

$$\alpha^* \approx 0.44(1 - \lambda)^2. \tag{8.14}$$

Since the theoretical results [NOS16] depends on the value of n, Fig. 8.5 shows different curves, corresponding to theoretical bounds on the convergence rates for different numbers of agents n. All these curves also present two regimes, however, the decrease of the first regime is slower and the sharp increase takes place after a much lower threshold step-size. Moreover, both the sharp increase and the threshold step-size worsen as n increases.

Therefore, the spectral rates obtained with PEP allow for significant improvements in the tuning of the step-size α by choosing a larger value (α^*), which is independent of n. This leads to better convergence guarantees and

therefore to better use of DIGing. For example, in the setting of Fig. 8.5, when $n = 20$, the theoretical bound requires a step-size below 10^{-4}, while the optimal step-size according to our spectral rate is around 4×10^{-3}. This choice of the step-size allows improving the convergence guarantee by at least 2 orders of magnitude.

We observed the same two regimes for all the other values of μ and λ we tested. However, the smaller the value of λ is, the larger the gap between the values of the theoretical and spectral convergence rate at the threshold step-size.

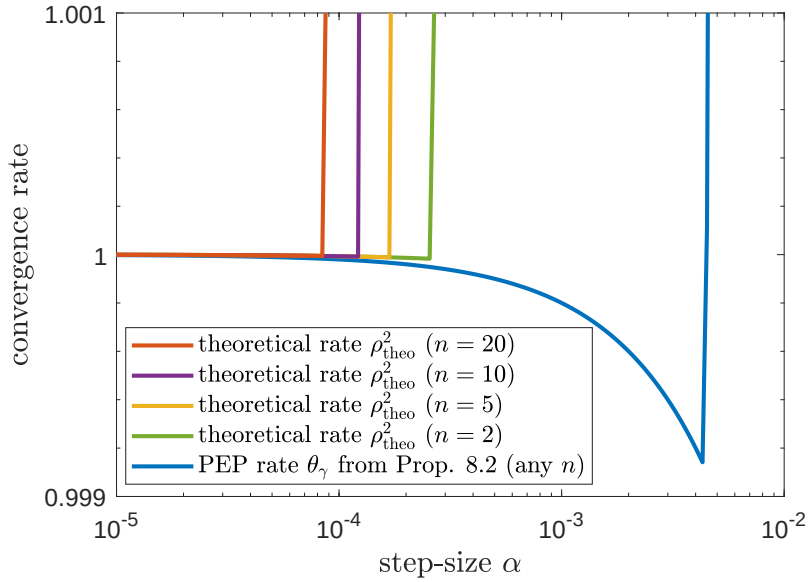


Fig. 8.5 Convergence rate evolution with the step-size α for DIGing with $\lambda = 0.9$ and $\mu = 0.1$. Theoretical rates from [NOS16] are shown for different values of n. PEP rates θ_γ, obtained with Proposition 8.2 for $\gamma = \frac{\alpha}{L}$, are lower and allow for larger step-sizes. These rates are computed with $n = 2$ but the results are valid for any value of n.

8.2.5 Impact of time-varying averaging matrices

In the spectral PEP formulation, we can choose to link different iterations together to analyze the worst-case when we use the same constant averaging matrix $W \in \mathcal{W}_{(-\lambda,\lambda)}$ at each iteration. We can also choose to consider each consensus step independently with potentially time-varying averaging matrices. For DIGing, we have chosen the second option to allow time-varying averaging matrices and be in the same condition as [NOS16]. However, we observe that both choices lead to the same worst-case values, even though the solution achieving these worst-cases may be different. One worst-case solution obtained with a constant matrix is therefore also a worst-case solution of the situation with time-varying matrices. We can thus analyze what is the worst-case constant matrix for DIGing and it will also be valid for time-varying settings.

8.2.6 Worst-case averaging matrix and tightness analysis

We follow the reasoning of Section 4.3.3 to check the tightness of the obtained PEP rates a posteriori, by recovering a valid worst-case averaging matrix. Tightness is double-checked using the exact agent-dependent PEP formulation with this worst-case averaging matrix. When the step-size α is optimized, i.e., equal to the threshold value α^*, we observe that the spectral PEP solutions for DIGing allow recovering a valid worst-case averaging matrix, which is the same as for DGD, i.e., $\hat{W}^* \in \mathcal{W}_{(-\lambda,\lambda)}$ from (8.2), see details in Section 4.5.2. This matrix is only determined by the value of λ and n and the remainder it produces $\|\tilde{Y} - \hat{W}^*\tilde{X}\|$ is always close to zero in the described settings, with $\alpha = \alpha^*$. The same worst-case matrix is also recovered for larger step-size α. The bounds obtained using the exact PEP formulation with this specific matrix $\hat{W}^*$ *exactly* match the corresponding spectral bounds, within numerical errors. This means that the spectral PEP formulation, even though it is a relaxation, provides a *tight performance bound* for DIGing with symmetric generalized doubly stochastic matrices and sufficiently large step-size.

In summary Our spectral PEP formulation provides numerical tight convergence rates for DIGing that are independent of the number of agents n and allows for better tuning of the constant step-size α, leading to more efficient use of the DIGing algorithm.

8.3 Accelerated Distributed Nesterov Gradient Descent

We now analyze the accelerated distributed Nesterov gradient descent (Acc-DNGD) algorithm proposed in [QL19]. We focus on the version designed for convex (not necessarily strongly convex) and L-smooth functions, described in Algorithm 4. It achieves one of the best convergence rates proven in such setting, i.e. $\mathcal{O}\left(\frac{1}{t^{1.4-\epsilon}}\right)$ for any $\epsilon \in (0, 1.4)$, but there remain several open questions on the choice of parameters and actual performance. We show how our PEP techniques allow shedding light on these questions. To maintain consistent notations throughout the thesis, we use notations that are slightly different from those in the original paper [QL19]. Here, α_k denotes the diminishing step-size and θ_k denotes a weighting factor. Each agent i keeps variables x_i, v_i, y_i, and s_i. The variables s_i are local gradient tracking variables allowing each agent to estimate the average gradient $\frac{1}{n}\sum_{i=1}^n \nabla f_i(y_i)$. The step-size α_k is diminishing as

$$\alpha_k = \frac{\alpha}{(k+k_0)^\beta}, \tag{8.15}$$

where $\alpha \in \left(0, \frac{1}{L}\right)$, $\beta \in (0, 2)$ and $k_0 \geq 1$. The sequence of θ_k starts with $\theta_0 = \sqrt{\alpha_0 L} \in (0, 1)$ and the next element of the sequence is each time computed as the unique solution in $(0, 1)$ of

$$\theta_{k+1}^2 = \frac{\alpha_{k+1}}{\alpha_k}(1-\theta_{k+1})\theta_k^2.$$

Algorithm 4 Acc-DNGD

Input: averaging matrix $W \in \mathbb{R}^{n\times n}$ and sequences α_k and θ_k,
Initialize $x_i^0 = v_i^0 = y_i^0 = 0$ and $s_i^0 = \nabla f(0)$ for all i,
For $k = 0, 1, \ldots$ and $i = 1, \ldots, n$,

$$\begin{aligned}
x_i^{k+1} &= \sum_{j=1}^n w_{ij} y_j^k - \alpha_k s_i^k, \\
v_i^{k+1} &= \sum_{j=1}^n w_{ij} v_j^k - \frac{\alpha_k}{\theta_k} s_i^k, \\
y_i^{k+1} &= \theta_{k+1} x_i^{k+1} + (1-\theta_{k+1}) v_i^{k+1}, \\
s_i^{k+1} &= \sum_{j=1}^n w_{ij} s_j^k + \nabla f_i(y_i^{k+1}) - \nabla f_i(y_i^k),
\end{aligned}$$

8.3.1 Theoretical performance guarantee from [QL19]

The convergence result [QL19, Theorem 4] guarantees that the algorithm achieves an average functional error bounded as

$$f(\overline{x}^t) - f(x^*) \leq \mathcal{O}\Big(\frac{1}{t^{2-\beta}}\Big) \qquad \text{for } \beta \in (0.6, 2).$$

Recall that $f(x) = \frac{1}{n}\sum_{i=1}^n f_i(x)$, $\overline{x}^t = \frac{1}{n}\sum_{i=1}^n x_i^t$ and x^* is a finite minimizer of f. This convergence guarantee for Acc-DNGD only holds under specific conditions concerning the values of α and k_0 [QL19, Theorem 4]. These assumptions seem strong since they impose, in particular, that α tends to 0 and k_0 tends to ∞ both when $\lambda(W)$ tends to 1 (disconnected graph) and to 0 (fully connected graph). As detailed in Section 2.3.2, $\lambda(W)$ denotes the second-largest eigenvalue modulus of the averaging matrix W and is related to the connectivity of the underlying graph.

Conjectures The authors of [QL19] conjecture that

(i) Exact values of parameters α, k_0 do not actually matter and we can choose values that are not satisfying assumptions from [QL19, Theorem 4], and still obtain a rate of $\mathcal{O}(\frac{1}{t^{2-\beta}})$. For example, authors of [QL19] used $\alpha = \frac{1}{2L}$, $k_0 = 1$ in their numerical experiments with $\beta = 0.61$.

(ii) Choosing $\beta \in [0, 0.6]$ leads to the same rate $\mathcal{O}(\frac{1}{t^{2-\beta}})$. The case $\beta = 0$ uses constant step-size α and is important because it would lead to a rate $\mathcal{O}(\frac{1}{t^2})$, i.e., the best possible rate in a centralized framework for similar settings [Nes18, Theorem 2.1.7].

(iii) The uncoordinated initial condition $s_i^0 = \nabla f_i(y_i^0)$ leads to the same convergence rates for the algorithm as the condition $s_i(0) = \nabla f(0) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(0)$ used in the proof in [QL19] and which, in practice, requires agents to perform an initial consensus averaging protocol.

8.3.2 Performance analysis with PEP

The described performance evaluation settings satisfy conditions from Theorem 6.13 (part 2), so that we know the resulting worst-case performance is independent of the number of agents. Therefore, as explained in Chapter 7, we use the spectral agent-dependent PEP formulation with

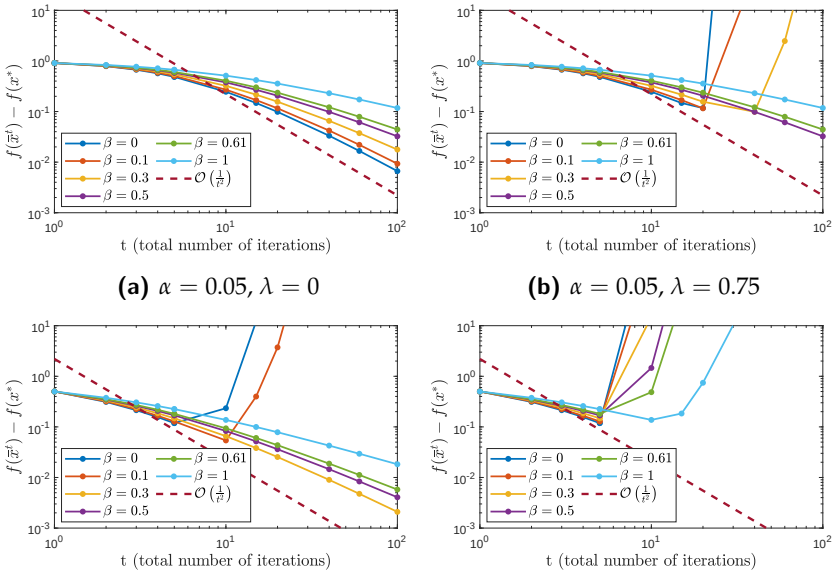
$n = 2$ agents to analyze Acc-DNGD in these settings. By Corollary 6.14, the obtained performance is valid for any number of agents $n \geq 2$. This PEP analysis will allow us to get a better idea of the impact of the parameters on the algorithm performance and to nuance the conjectures of [QL19].

Fig. 8.6 shows the evolution of the PEP worst-case average functional error $f(\overline{x}^t) - f(x^*)$ with the total number of iterations t. We consider a symmetric range of eigenvalues $-\lambda^- = \lambda^+ = \lambda$, and thus we obtain spectral bounds valid over the entire class of averaging matrices $\mathcal{W}_{(-\lambda,\lambda)}$. The results are shown for different values of β, α and λ, while L, n and k_0 are fixed. Fig. 8.6 aims at testing the conjectures (i) and (ii). To test (ii), about the validity range of β, conjectured to extend below 0.6, we choose $\beta = \{0, 0.1, 0.3, 0.5\}$. To compare with proven valid values of β and test conjecture (i), we also choose $\beta = \{0.61, 1\}$. We choose $\lambda = \{0, 0.75\}$ and $\alpha = \{0.05, 0.5\}$ to have two representative values for each parameter (small and large). We fix $L = 1$, because the results for other values of L can be recovered by scaling, and $k_0 = 1$ for simplicity and because it does not affect the long-term evolution of the performance. In particular, the value of k_0 has no impact when $\beta = 0$, see (8.15).

(i) Impact of the initial step-size α We observe that the value of α influences the performance of the algorithm. Indeed, when α becomes larger (e.g., from Fig. 8.6a, 8.6b to Fig. 8.6c, 8.6d), the worst-case functional error decreases faster in a first phase but it sometimes explodes afterward, preventing the algorithm to converge. This occurs for example when β is too small (e.g., $\beta = 0$ and $\beta = 0.1$ in Fig. 8.6c) or when λ is too large (e.g., in Fig. 8.6d and 8.6b where $\lambda = 0.75$). In Fig. 8.6d, this sharp increase even occurs in the experimental setting of [QL19], i.e., $\alpha = \frac{1}{2L} = 0.5$ and $\beta = 0.61$ (green line). These observations contradict conjecture (i) as currently stated.

(ii) Small values for β We observe in Fig. 8.6 that the curves for $\beta = 0.5$ and $\beta = 0.61$ behave similarly in all the different plots. This suggests that the worst-case values present no phase transition at $\beta = 0.6$, and therefore, supports conjecture (ii) claiming that values of β lower than 0.6 also provide rates $\mathcal{O}(\frac{1}{t^{2-\beta}})$, including the case $\beta = 0$. The curves for $\beta = 0$ appear indeed to approach a decrease in $\mathcal{O}(\frac{1}{t^2})$, as conjectured in [QL19], but only in Fig. 8.6a where the values α and λ are sufficiently small. Indeed, the rate $\mathcal{O}(\frac{1}{t^2})$ cannot be observed for larger values of α or λ where PEP problems

may become unbounded (see Figures 8.6b, 8.6c and 8.6d). Therefore, the α parameter should be tuned according to the values of λ and the choice of β. Qualitatively, we can see that it must decrease when λ increases or when β decreases. See Fig. 8.7 for tuning α with $\beta = 0$, $t = 10$ and $\lambda = 0$ or $\lambda = 0.75$. Further analysis should be performed to tune the value of α in general. Having sufficiently small step-sizes α seems thus to be the key for conjecture (ii) to be true. Since smaller values for β appear to require a smaller step-size, the performance would approach the convergence rate $\mathcal{O}(\frac{1}{t^{2-\beta}})$ more slowly in that case. However, even with well-tuned α, we cannot exclude that the worst-case performance of Acc-DNGD does not explode after a large number of iterations in some cases, as is the case in an early phase for some settings in Fig. 8.6.



(a) $\alpha = 0.05, \lambda = 0$ **(b)** $\alpha = 0.05, \lambda = 0.75$

(c) $\alpha = 0.5, \lambda = 0$ **(d)** $\alpha = 0.5, \lambda = 0.75$

Fig. 8.6 Evolution with t of the worst-case average functional error $f(\overline{x}^t) - f(x^*)$ for Acc-DNGD, obtained with the spectral PEP formulation, with $n = 2$, $L = 1$ and $k_0 = 1$. Different values of β, α and λ are shown. Dashed lines show curves evolving in $\mathcal{O}(\frac{1}{t^2})$, which corresponds to the rate conjectured in [QL19] when $\beta = 0$. This asymptotic rate is reached for $\beta = 0$ only when α is sufficiently small.

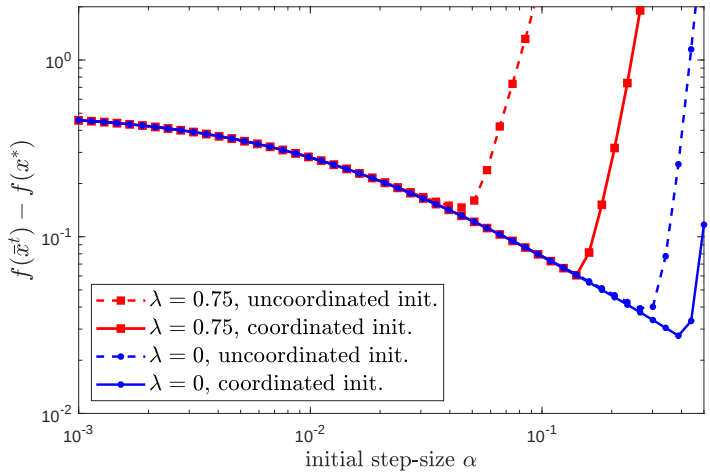


Fig. 8.7 Evolution with α of the worst-case performance of $t = 10$ iterations of Acc-DNGD, obtained with the spectral PEP formulation, with $n = 2$, $L = 1$, $k_0 = 1$ and $\beta = 0$. The plot compares the coordinated initialization described in Algorithm 4 and the uncoordinated initialization (8.16)-(8.17) for $\lambda = 0$ and $\lambda = 0.75$.

(iii) Initial conditions Fig. 8.7 aims at testing conjecture (iii) about the initial conditions used for the algorithm. The figure shows the evolution of the worst-case performance of Acc-DNGD with parameter α, both for the coordinated initial conditions described in Algorithm 4 and for the uncoordinated initial conditions (8.16)-(8.17), described below:

$$x_i^0 = v_i^0 = y_i^0 \in \mathbb{R}^d, \quad \text{and} \quad s_i^0 = \nabla f_i(y_i^0), \tag{8.16}$$

$$\frac{1}{n}\sum_{i=1}^{n} \|x_i^0 - x^*\|^2 \leq 1, \quad \text{and} \quad \frac{1}{n}\sum_{i=1}^{n} \|s_i^0 - \frac{1}{n}\sum_{j=1}^{n} \nabla f_j(y_j^0)\|^2 \leq 1. \tag{8.17}$$

These conditions are less restrictive than those in Algorithm 4 and do not require initial coordination between agents. They are similar to the conditions used in the DIGing algorithm, see Section 8.2. The inequalities (8.17) involve initial points from all the agents, but they are only used in the performance analysis of the algorithm and not by agents to determine their initial points in practice. In Fig. 8.7, we observe that the different values of λ and the different initializations all lead to the same decay phase with α, but that they differ in the threshold value α_t at which the performance starts to increase with α. Therefore, these different settings lead to the same

worst-case performance for α sufficiently small, but have different optimal values for the choice of parameter α. Uncoordinated initial conditions require a smaller value for α than the coordinated ones. Moreover, large values for λ also require a smaller value for α. We conclude that conjecture (iii) seems to hold for values of α sufficiently small. However, we reveal that the initialization impacts the tuning of parameter α.

Tightness analysis The worst-case averaging matrix that is recovered from the spectral PEP formulation for Acc-DNGD is again $\hat{W}^*$ from (8.2), as it was the case for DGD and DIGing. For Acc-DNGD also, the bounds obtained using the exact PEP formulation with this specific matrix $\hat{W}^*$ *exactly* match the corresponding spectral bounds, within numerical errors. This means that the spectral PEP formulation, even though it is a relaxation, provides a *tight performance bound* for Acc-DNGD with symmetric generalized doubly stochastic matrices.

8.3.3 Recent theoretical results from [LL21]

The recent preprint [LL21] proposes a reformulation of Algorithm 4 with constant step-size α, corresponding to the case $\beta = 0$ in (8.15). The algorithm is called Acc-GT. For smooth convex functions, it differs from Acc-DNGD only in the update order and notations. The initialization also requires an initial coordination step, to ensure all the agents start with the same points. However, Acc-GT extends differently than Acc-DNGD to strongly convex functions. In [LL21], the authors provide a refined theoretical analysis of the algorithm and prove a convergence in $\mathcal{O}(\frac{1}{t^2})$ for smooth convex functions, which confirms conjecture (ii) for $\beta = 0$. Moreover, their analysis also provides dependence to the problem parameters (L and λ), so that we know the algorithm requires $\mathcal{O}(\frac{1}{(1-\lambda)^2}\sqrt{\frac{L}{\epsilon}})$ gradients computation and rounds of communications to find a solution such that $f(\overline{x}^k) - f(x^*) \leq \epsilon$. This complexity has the optimal dependence on the precision ϵ and the smoothness parameter L, matching that of the classical centralized accelerated gradient method (see Table 2.1). However, the dependence on the network connectivity $(1-\lambda)$ is not optimal. This dependence can be improved by considering an inner loop with multiple consensus steps, such as the Chebyshev acceleration, when the averaging matrix is time-invariant, see [LL21] for details. Finally, in [LL21], the authors provide similar convergence guarantees when the three consensus steps involved in the updates use different averaging matrices, and also when

these averaging matrices are time-varying. While these results represent major progresses, they require very specific and small step-sizes, that we suspect to be conservative. Our PEP analysis can be efficient in determining wider ranges of validity for the convergence results.

8.4 EXTRA

In this section, we leverage our compact PEP formulation, developed in Chapter 6 by using equivalence classes of agents, to evaluate the performance of the EXTRA algorithm [SLWY14] in advanced settings, and the performance evolution with system size n. This provides totally new insights into the worst-case performance of the algorithm. The EXTRA method, described in Algorithm 2, is a well-known distributed optimization method that was introduced in [SLWY14] and further developed in [SLWY15, Jak18, LL20], see Section 2.4 for more details.

Theoretical performance guarantees from [LL20] The sharpest convergence results for EXTRA, with $\tilde{W} = \frac{W+I}{2}$, are given in [LL20], and summarized below for the strongly convex case.

Theorem 8.3 (Theoretical performance guarantee for EXTRA [LL20])**.** *We consider the decentralized optimization problem* (2.1) *with optimal solution* $x^* \in \mathbb{R}^d$*. Under the following assumptions,*

1. *All local functions are L-smooth and* μ*-strongly convex, with* $0 \leq \mu \leq L$*,*
$$f_i \in \mathcal{F}_{\mu,L} \qquad \text{for all } i = 1, \dots, n;$$
2. *Averaging matrix* $W \in \mathcal{W}_{(-\lambda,\lambda)}$ (4.10)*, with* $\lambda \in [0, 1)$*, and* $\tilde{W} = \frac{W+I}{2}$*;*
3. *The starting points satisfy, for some constant* $D \in \mathbb{R}, R \in \mathbb{R}$*,*
$$\|x_i^0 - x^*\|^2 \leq D^2 \qquad \text{and} \qquad \|\nabla f_i(x^*)\|^2 \leq R^2 \qquad \text{for all } i = 1, \dots, n;$$

the EXTRA algorithm with $\alpha = \frac{1}{4L}$ *guarantees that*

$$P_f(t) := f(\overline{x}^t) - f(x^*) \leq (1-\tau)^t \, \frac{LD^2 + R^2/L}{1-\lambda}, \qquad \textit{(function error)} \quad (8.18)$$

$$P_x(t) := \frac{1}{n} \sum_{i=1}^{n} \|x_i^t - x^*\|^2 \leq (1-\tau)^t \, \frac{D^2 + R^2/L^2}{1-\lambda}, \qquad \textit{(iterate error)} \quad (8.19)$$

where $\overline{x}^t = \frac{1}{n}\sum_{i=1}^n x_i^t$ *and* $\tau = \frac{1}{39\left(\frac{L}{\mu} + \frac{1}{1-\lambda}\right)}$.

Proof. Bounds (8.18) and (8.19) directly follow from [LL20, Corollary 1]. □

For comparison purposes, our PEP-based analysis of EXTRA will use the same assumptions as in Theorem 8.3, with $L = 1$, $\mu = 0.1$, $\lambda = 0.5$, $D = 1$ and $R = 1$, unless otherwise specified. Our compact PEP formulation, with equivalence classes of agents (Chapter 6), also enables the analysis of a variety of settings other than those of Theorem 8.3, many of which have not yet been studied in the literature. In this section, we explore, in particular, the performance of the worst agent, the k-th percentile performance, and the performance under agent heterogeneity in the classes of local functions. Other possible settings include agent heterogeneity in the initial conditions, the network topology, the algorithm parameters, or algorithm execution. For example, it could be used to analyze the performance of an algorithm when there is a subset of malicious agents.

8.4.1 Performance of the worst agent

Classical analysis of distributed optimization algorithms often focuses on performance metrics averaged over all the agents, namely

$$P_f(t) = f(\overline{x}^t) - f(x^*) \qquad \text{or} \qquad P_x(t) = \frac{1}{n}\sum_{i=1}^{n} \|x_i^t - x^*\|^2, \quad (8.20)$$

where $\overline{x}^t = \frac{1}{n}\sum_{i=1}^n x_i^t$. Theoretical guarantees on these criteria for EXTRA are given in (8.18) and (8.19) from Theorem 8.3. As we have seen in Chapter 6, these error criteria are scale-invariant and lead to performance bounds that are independent of the number of agents n (see Theorem 6.13) and are therefore easier to treat analytically. However, there exist other relevant performance measures that are more difficult to analyze and for which there are few or no results. This is the case of the performance of the worst agent in the network,

$$P_{f,\text{worst}}(t) = \max_{i\in\mathcal{V}} \; f(x_i^t) - f(x^*) \quad (8.21)$$

$$\text{or} \quad P_{x,\text{worst}}(t) = \max_{i\in\mathcal{V}} \; \|x_i^t - x^*\|^2, \quad (8.22)$$

where $\mathcal{V} = \{1, \dots, n\}$ is the set of agents in the network. To analyze these performance measures using PEP, we need to fix (arbitrarily) the agent that will be the worst one and use it in the PEP objective:

$$f(x_1^t) - f(x^*) \qquad \text{or} \qquad \|x_1^t - x^*\|^2,$$

The maximization of such objectives ensures that agent 1 will be the one with the largest error (on f or x). Indeed, if another agent j has a larger er-

ror, then the permutation of variables of agents 1 and j in the PEP solution would lead to another feasible PEP solution with a larger objective. Agent 1 will thus receive a specific role in the PEP solution and is not equivalent to the $n-1$ others. We can formulate a compact PEP using two equivalence classes of agents

$$\mathcal{V}_1 = \{1\} \qquad \text{and} \qquad \mathcal{V}_2 = \{2, \dots, n\},$$

and applying the theory developed in Section 6.3.

The worst agent functions error Figure 8.8 shows the evolution of P_f and $P_{f,\text{worst}}$ with the size of the network, for 15 iterations of the EXTRA algorithm. These results have been computed with the compact PEP formulation (from Chapter 6) and validated with the agent-dependent PEP formulation (from Chapter 4) for small values of n, for verification purposes. However, as n becomes larger, this agent-dependent formulation becomes computationally intractable, reaching an SDP size of 325×325 for $n = 10$ (and $t = 15$). This limitation precisely motivated the development of the compact symmetrized PEP formulations from Chapter 6.

Firstly, Figure 8.8 shows that the PEP bound for the average performance P_f is well constant over n, as predicted by Theorem 6.13, and outperforms the theoretical bound from [LL20, Corollary 2], which requires 2750 iterations to guarantee the same error as the PEP bounds after 15 iterations. The theoretical bound is valid for a step-size $\alpha = \frac{1}{4L}$, while the PEP bound is shown for an optimized step-size $\alpha^* = \frac{0.78}{L}$, which minimizes the bound. Since the bound is constant over n, the value of the optimized step-size can be used for any system size, however, it may depend on other settings, such as the range of eigenvalues for W: $[\lambda^-, \lambda^+]$.

Then, Figure 8.8 also shows that the PEP bound for the worst agent $P_{f,\text{worst}}$ (in red) is scaling sublinearly with the number of agents n. Using the constant step-size $\alpha^* = \frac{0.78}{L}$ (plain red line), we obtain a logarithmic slope of 0.92. The step-size can also be tuned with respect to $P_{f,\text{worst}}$, and would then have a different value for each value of n, denoted $\alpha^*_{\text{w}}(n)$. Using $\alpha^*_{\text{w}}(n)$ (dashed red line), we obtain a logarithmic slope of 0.67, which means the scaling with n is less important when using suitable step-sizes. In our case, the step-sizes $\alpha^*_{\text{w}}(n)$ are diminishing in $\frac{1}{\sqrt{n}}$. However, these smaller step-sizes deteriorate the average performance P_f (dashed blue line). There is currently no theoretical guarantee for the worst agent functions error $P_{f,\text{worst}}$ (8.21) of EXTRA to compare with, and our analysis is pioneering in this field.

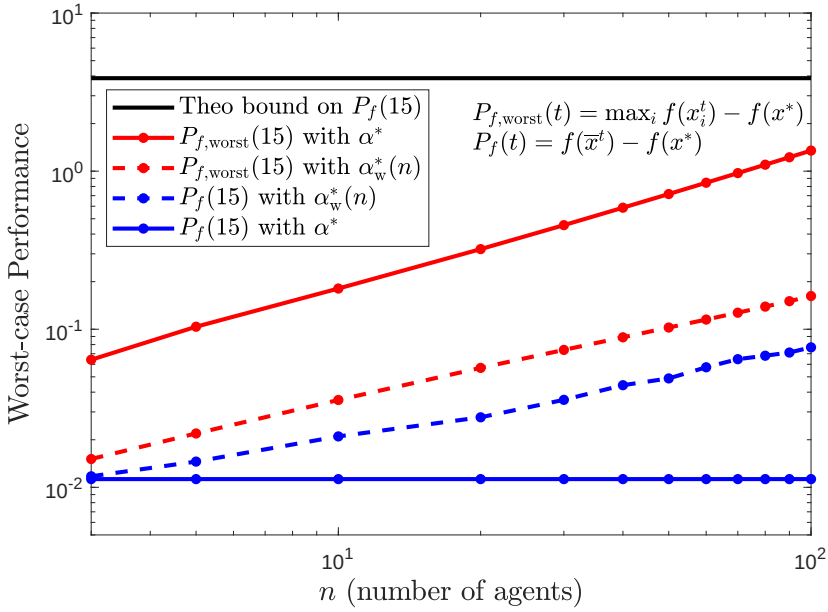


Fig. 8.8 Comparison of the average functional error $P_f(t)$ (8.20) and the worst functional error $P_{f,\text{worst}}(t)$ (8.21) for $t = 15$ iteration of the EXTRA algorithm and their evolution with the number of agents n in the system. The plot shows (i) the constant theoretical bound on P_f (8.18), in black, (ii) the PEP bounds on the function error of the worst agent $P_{f,\text{worst}}$, scaling sublinearly with n, in red and (iii) the PEP bounds for the function error of the average iterate P_f, in blue. The constant step-size α^* has been optimized for the performance criterion $P_f(15)$ and leads to a bound constant with n (plain blue line). Step-sizes can also be optimized with respect to $P_{f,\text{worst}}(15)$, leading to smaller step-sizes $\alpha^*_{\text{w}}(n)$ decreasing as $\frac{1}{\sqrt{n}}$. They improve the guarantee for $P_{f,\text{worst}}$ and its scaling with n but they deteriorate the corresponding guarantee for P_f (see dashed lines). Parameter values: $L = 1$, $\mu = 0.1$, and $\lambda = 0.5$.

The worst agent iterate error There is currently no theoretical guarantee for the worst agent iterate error $P_{x,\text{worst}}$ (8.22) of EXTRA in the literature, but one could derive[4] a conservative bound on $P_{x,\text{worst}}$ from (8.19), scaling linearly with the number of agents n. Using the compact PEP formulation from Chapter 6, we can analyze such performance metric accurately and show that the performance of the worst agent $P_{x,\text{worst}}$ for EXTRA actually scales sublinearly with n, as shown in Figure 8.9. Figure 8.9 shows PEP-based bounds for P_x and $P_{x,\text{worst}}$. The observations are similar to those of Figure 8.8. The PEP bound for the average performance P_x is constant over n, as predicted by Theorem 6.13 and the corresponding optimized step-size is $\alpha^* = 0.78$, for $L = 1$ and $\mu = 0.1$. Moreover, the PEP bound for the worst agent $P_{x,\text{worst}}$ is scaling sublinearly with the number of agents n. Using the constant step-size $\alpha^* = 0.78$ (plain red line), we obtain a logarithmic slope of 0.82. As previously, the step-size can also be tuned with respect to $P_{x,\text{worst}}$, and would then have a different value for each value of n, denoted $\alpha^*_{\text{w}}(n)$. Using the optimal diminishing step-sizes $\alpha^*_{\text{w}}(n)$ (dashed red line), we obtain a logarithmic slope of 0.60, which means the scaling with n is less important when using suitable step-sizes. These smaller step-sizes only slightly deteriorate the average performance P_x (dashed blue line).

8.4.2 The 80-th percentile of the agent performance

In some cases, the performance of the worst agent may not be the most appropriate performance measure. When the network is very large, the performance of the worst agent can be terribly bad, while the average performance is good. Using our PEP approach, we can consider other performance measures that were so far completely out of reach of theoretical analysis. Inspired by statistical approaches, we propose here to analyze the k-th percentile in the distribution of the individual performance of each agent. In this experiment we have chosen the 80-th percentile, that is the performance of the worst agent, after excluding the 20% worst agents. In other words, it measures the error at or below which 80% of the agents fall in the worst-case scenario. The performance of the worst agent, analyzed in the previous subsection, can be seen as the 100-th percentile of the agent performance and can be analyzed via PEP using two equivalence classes of agents.

[4] Using the fact that $\max_{i\in\mathcal{V}} \|x_i^t - x^*\|^2 \leq \sum_{i=1}^n \|x_i^t - x^*\|^2$.

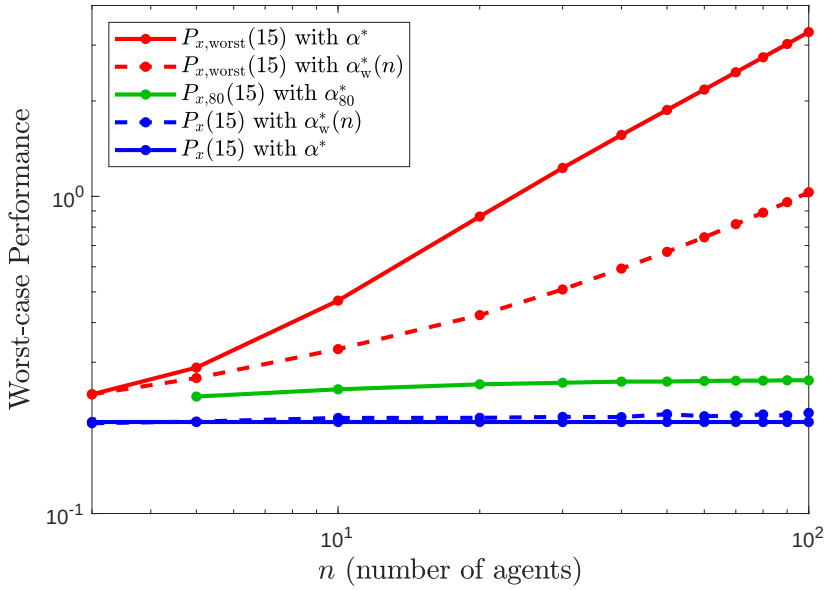


Fig. 8.9 Comparison of the average iterate error $P_x(t)$ (8.20), the worst iterate error $P_{x,\mathrm{worst}}(t)$ (8.22) and the 80-th percentile iterate error $P_{x,80}(t)$ (8.23) for $t = 15$ iterations of the EXTRA algorithm and their evolution with the number of agents n in the system. The plot shows (i) the PEP bounds on the iterate error of the worst agent $P_{x,\mathrm{worst}}$, scaling sublinearly with n, in red, (ii) the PEP bound on the 80-th percentile iterate error $P_{x,80}$, in green, and (iii) the PEP bounds for the average iterate error P_x, in blue. The constant step-size α^* has been hand-tuned for the performance criterion $P_x(15)$ and leads to a bound constant with n (plain blue line). Step-sizes can also be hand-tuned with respect to $P_{x,\mathrm{worst}}(15)$, leading to smaller step-sizes $\alpha^*_{\mathrm{w}}(n)$. They improve the guarantee for $P_{x,\mathrm{worst}}$ and its scaling with n without deteriorating too much the corresponding guarantee for P_x (see dashed lines). Parameter values: $L = 1$, $\mu = 0.1$, and $\lambda = 0.5$.

When considering smaller percentiles, such as the 80-th, we can formulate a compact PEP using three equivalence classes of agents:

$$\mathcal{V}_1 = \{1, \dots, 0.2n\}, \qquad \mathcal{V}_2 = \{0.2n+1\}, \qquad \mathcal{V}_3 = \{0.2n+2, \dots, n\},$$

where we assume n can be divided by 5. The class $\mathcal{V}_1$ contains the 20% worst agents to exclude, the class $\mathcal{V}_2$ contains the agent for which the individual performance will provide the 80-th percentile, and $\mathcal{V}_3$ contains all the other agents. The PEP will therefore maximize the performance of agent $i \in \mathcal{V}_2$ and imposes that the performance of each agent $j \in \mathcal{V}_1$ is larger:

$$\begin{aligned} P_{x,80}(t) = \underset{\{f_A^u, G_A^u, G_B^u G_E^{uv}\}_{u,v=1,2,3}}{\text{maximum}} \quad & \|x_i^t - x^*\|^2 \quad \text{with } i \in \mathcal{V}_2 \\ \text{s.t.} \quad & \|x_j^t - x^*\|^2 \geq \|x_i^t - x^*\|^2 \quad \text{for all } j \in \mathcal{V}_1, \\ & \text{All the other PEP constraints.} \end{aligned} \tag{8.23}$$

In problem (8.23), the squared norms $\|x_i^t - x^*\|^2$ can be written as $(e_{x^t} - e_{x^*})^T G_A^u (e_{x^t} - e_{x^*})$ with u chosen according to the class $\mathcal{V}_u$ containing agent i. Details about the explicit construction of compact PEP formulations are given in Appendix 6.A.

Figure 8.9 shows the evolution of $P_{x,80}$ with the size of the network, for 15 iterations of the EXTRA algorithm (green line). We observe that the scaling of $P_{x,80}$ with the number of agents n is very limited and quickly reaches a plateau, making this performance almost as good as the average performance P_x. Moreover, the optimal step-size for this bound, denoted α_{80}^*, is independent of n and, in this case, it stays close to α^*, optimized for P_x. The value of the plateau for $P_{x,80}$ has been confirmed by solving the compact PEP with n tending to ∞, as described at the end of Section 6.3.

Figure 8.10 shows the evolution with k of the k-th percentile of the agent performance for EXTRA, when $n \to \infty$. We see that the limit of the k-th percentile performance is finite for all values of $k \in (0, 100)$ and has a vertical asymptote in $k = 100$. The 100-th percentile computes the performance of the worst agent and grows well to infinity with n (see Figure 8.9). According to Figure 8.10, the value of the k-th percentile performance first increases slowly with k, and then it starts to blow up after some threshold on k, depending on t, to match the vertical asymptote in $k = 100$. This means that it seems useful to exclude a small part of the worst agents in

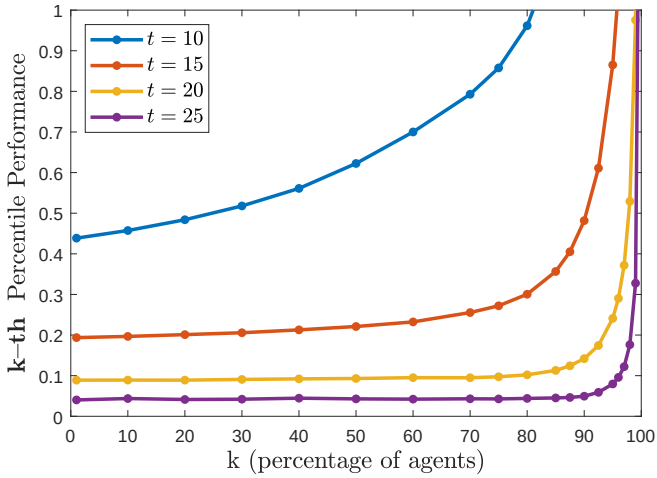


Fig. 8.10 Evolution with k of the k-th percentile of the agent performance for t iterations of EXTRA, when the number of agents tends to infinity ($n \to \infty$). Different total numbers of iterations t are compared. Parameter values: step-size $\alpha = \alpha^* = 0.78$, $L = 1$, $\mu = 0.1$, and $\lambda = 0.5$.

the performance metric, e.g., 10–20% in the settings we are considering, but not more. Moreover, the k-th percentile of EXTRA for $n \to \infty$ seems to converge geometrically. Indeed, in Figure 8.10, when considering 5 more iterations in total, the k-th percentile improves by a factor between 2.2 and 3.2, depending on k.

Remark 8.4. Figures 8.9 and 8.10 show the performance of EXTRA for rather well-connected averaging matrices ($\lambda = 0.5$). Similar results and observations can be obtained for larger ranges of eigenvalues, i.e., $\lambda \in (0.5, 1)$, but this would require more iterations to get small errors.

8.4.3 Performance under local functions heterogeneity

In many applications, the agents hold local functions sharing general properties, e.g., convexity or smoothness, but it is unlikely that all the local functions have exactly the same characterization for these properties. For example, the local functions can be μ_i-strongly convex and L_i-smooth, but with different parameters μ_i and L_i for each i. In general, the performance is then computed by considering that all the local functions have the worst

parameters:

$$\mu = \min_i \mu_i \qquad \text{and} \qquad L = \max_i L_i.$$

In this experiment, we analyze the gain in the performance guarantee when instead of considering $f_i \in \mathcal{F}_{\mu,L}$ for all i, we consider two equivalence classes of agents $\mathcal{V}_1$ and $\mathcal{V}_2$ such that

$$f_i \in \mathcal{F}_{\mu_1,L} \quad \text{for all } i \in \mathcal{V}_1 \quad \text{and} \quad f_i \in \mathcal{F}_{\mu_2,L} \quad \text{for all } i \in \mathcal{V}_2.$$

These two classes of agents manipulate functions with two different condition numbers $\kappa_1 = \frac{L}{\mu_1}$ and $\kappa_2 = \frac{L}{\mu_2}$. Figure 8.11 shows the situation where $L = 1$, $\mu_1 = 0.01$ and $\mu_2 = 0.1$, for different sizes of classes $\mathcal{V}_1$ and $\mathcal{V}_2$. We observe that the guarantees for EXTRA only depend on the relative composition of each class $\mathcal{V}_1$ and $\mathcal{V}_2$ but not on the total number

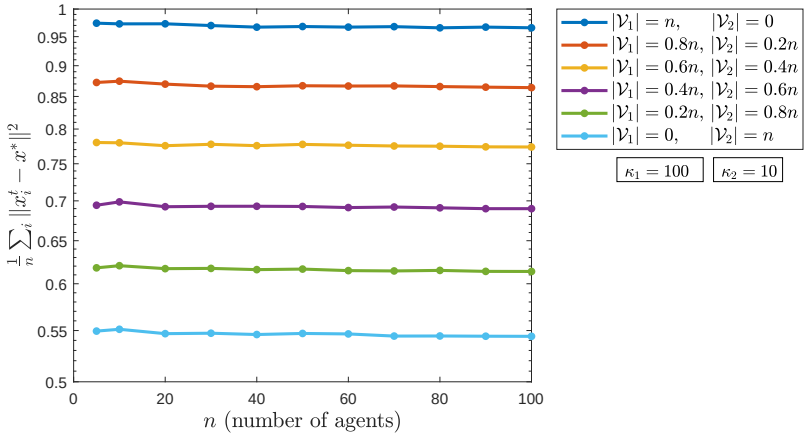


Fig. 8.11 Evolution with the number of agents n of the worst-case average distance to optimum $P_x(t)$ (8.20) for 15 iterations of EXTRA with two equivalence classes of agents $\mathcal{V}_1$ and $\mathcal{V}_2$ using each a different condition number for their function class. The agents in $\mathcal{V}_1$ hold 1-smooth and 0.01-strongly convex local functions ($\kappa_1 = 100$) and the agents in $\mathcal{V}_2$ hold 1-smooth and 0.1-strongly convex local functions ($\kappa_2 = 10$). The guarantees are independent of the total number of agents n and only depend on the proportion of agents in each class. In this experiment, we set the step-size to $\alpha = 0.21$, which is optimal for the case where all the agents are in $\mathcal{V}_1$ ($L = 1$, $\mu = 0.01$), and we set $\lambda = 0.5$.

of agents n. This observation has been confirmed by solving the compact PEPs for n tending to ∞ (see Section 6.3.2). In Figure 8.11, when 20% of the agents pass from the class $\mathcal{V}_1$ to the class $\mathcal{V}_2$, the worst-case guarantee is improved by a constant factor 0.9. This improving factor can be linked to the worst-case guarantees obtained with a uniform value for κ in the local functions ($\kappa = 100$ or $\kappa = 10$ for all the agents) and motivates conjecture 8.5 below. One hypothesis is that the important conditioning is the one of the global function $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$. When θn agents hold a local function in $\mathcal{F}_{\mu_1,L}$ and $(1-\theta)n$ agents hold one in $\mathcal{F}_{\mu_2,L}$, the global function $f(x)$ is in $\mathcal{F}_{\theta\mu_1+(1-\theta)\mu_2,L}$, by Definition 3.2 of $\mathcal{F}_{\mu,L}$ functions. Therefore, we have tested our hypothesis by comparing the performance of EXTRA under local functions heterogeneity and the performance when all the local functions are in $\mathcal{F}_{\theta\mu_1+(1-\theta)\mu_2,L}$. We have observed that both situations provide the same worst-case performance, which confirms that the important condition number is indeed that of the global function $f(x)$.

Conjecture 8.5. *Let $P_x^{\kappa_1,\kappa_2}(t,\theta)$ be the performance guarantee on the iterates average error (8.20) after t iterations of EXTRA with θn agents with local functions in $\mathcal{F}_{\mu_1,L}$ and $(1-\theta)n$ agents with local functions in $\mathcal{F}_{\mu_2,L}$, for $\theta \in [0,1]$. Let $P_x^{\kappa_1}(t)$, $P_x^{\kappa_2}(t)$, and $P_x^{\overline{\kappa}}(t)$ be the worst-case errors of t iterations of the EXTRA algorithm with uniform conditioning for all the local functions, respectively with $\kappa_1 = \frac{L}{\mu_1}$, $\kappa_2 = \frac{L}{\mu_2}$ and $\overline{\kappa} = \frac{L}{(\theta\mu_1+(1-\theta)\mu_2)}$. By definition, we have*

$$P_x^{\kappa_1}(t) = P_x^{\kappa_1,\kappa_2}(t,1) \quad \text{and } P_x^{\kappa_2}(t) = P_x^{\kappa_1,\kappa_2}(t,0)$$

We conjecture that the performance $P_x^{\kappa_1,\kappa_2}(t,\theta)$ is related to the performance with uniform conditioning $P_x^{\kappa_1}(t)$, $P_x^{\kappa_2}(t)$, and $P_x^{\overline{\kappa}}(t)$, as follows

$$P_x^{\kappa_1,\kappa_2}(t,\theta) = \left(P_x^{\kappa_1}(t)\right)^{\theta}\left(P_x^{\kappa_2}(t)\right)^{1-\theta}, \quad \text{for all } t>0,\ \kappa_1,\kappa_2>0 \text{ and } \theta\in[0,1],$$

$$P_x^{\kappa_1,\kappa_2}(t,\theta) = P_x^{\overline{\kappa}}(t), \qquad \text{for all } t>0,\ \kappa_1,\kappa_2>0 \text{ and } \theta\in[0,1].$$

In addition to the values of κ_1, κ_2 and θ shown in Figure 8.11, this conjecture has been verified for different other values of κ_1, κ_2 and θ. It can also be extended to any number of agent equivalence classes, associated with different values of κ. For example, we have verified it with 3 classes of agents. This results in much tighter bounds in cases where only a few agents in the system hold ill-conditioned local functions. As a future direction, this also allows exploring situations where the agents hold totally

different types of local functions, e.g., strongly convex ($\mu > 0$), convex ($\mu = 0$) and weakly convex ($\mu < 0$) functions, or smooth and non-smooth functions. This conjecture would have been difficult to guess without the PEP tool and the compact formulation developed in Chapter 6.

8.5 Comparison of algorithms

In this section, we compare different decentralized optimization algorithms on a fair basis, to know which ones are the best to use in which cases. We will see that a slight variant of an algorithm can lead to substantial improvement in its worst-case performance (like the ATC variant of DIGing). Moreover, we also show how the performance of the algorithms adapts if the averaging matrices are time-varying.

In the literature, the convergence analysis of algorithms applies to different performance criteria or initial conditions, making it difficult to compare them fairly. Moreover, the level of tightness can vary from one guarantee to the other, which can make the comparison meaningless. Our framework enables easy comparison of algorithms based on their worst-case performance PEP bounds. In this section, we propose to compare the performance of different algorithms over t iterations, in the following settings:

- **Performance measure**: the agent average of the last local iterate error,

$$\mathcal{P}_x(t) = \frac{1}{n}\sum_{i=1}^{n} \|x_i^t - x^*\|^2. \tag{8.24}$$

- **Class of functions**: $\mathcal{F}_{\mu,L}$, i.e., L-smooth and μ-strongly convex functions (Definition 3.2).

- **Initial conditions**: the initial iterates $x_1^0, \dots, x_n^0 \in \mathbb{R}^d$ satisfy

$$\frac{1}{n}\sum_{i=1}^{n} \|x_i^0 - x^*\|^2 \leq D^2,$$
$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(x_i^0) - \frac{1}{n}\sum_{j=1}^{n} \nabla f_j(x_j^0)\|^2 \leq E^2.$$

- **Class of averaging matrices**: $\mathcal{W}_{(-\lambda,\lambda)}$ for $\lambda \in [0,1)$, i.e., symmetric averaging matrices with eigenvalues $\lambda_2, \ldots, \lambda_n \in [-\lambda, \lambda]$, see (4.10).

These common settings satisfy conditions of Theorem 6.13, meaning that the worst-case bounds for algorithms from $\mathcal{A}_D$, with coordinated[5] parameters, are independent of the number of agents. In that case, as explained in Chapter 7, we use the spectral agent-dependent PEP formulation with $n = 2$, and we obtain worst-case bounds that are valid for all $n \geq 2$. Therefore, our comparisons are valid over the entire spectral class of networks $\mathcal{W}_{(-\lambda,\lambda)}$, for all network sizes $n \geq 2$.

We compare the following decentralized optimization algorithms:

- DIGing [NOS16], see Algorithm 3, with the optimized step-size $\alpha^* = 0.44(1-\lambda)^2$ (8.14).
- ATC-DIGing [NOSU17], see equations (2.14)-(2.15), with an ad-hoc optimized step-size α.
- EXTRA [SLWY14], see Algorithm 2, with $\tilde{W} = \frac{1}{2}(W + I)$ and an ad-hoc optimized step-size.
- Acc-DNGD-SC [QL19], which is a variant of Algorithm 4 with constant parameters and is suited to smooth and strongly convex functions. We used an ad-hoc optimized step-size α.
- Acc-GT [LL21], which is another variant of Algorithm 4 with constant parameters, is suited to smooth and strongly convex functions and has a better proven convergence rate than Acc-DNGD-SC (for the functional error). We used an ad-hoc optimized step-size α.

In our experiments, we set $D = 1$, $E = 1$, $\mu = 0.1$, and $L = 1$, but results for general values of D and L can be recovered by scaling.

Fig. 8.12 compares the algorithms in these settings, after $t = 10$ iterations, for different ranges of eigenvalues λ, and for both, time-invariant and time-varying averaging matrices.

- **Time-invariant averaging matrices**. It means that the same matrix is used for each consensus. As explained in Chapter 4, this is ensured in PEP by imposing conditions from Corollary 4.8 to all consensus steps at once.

[5] Equal for all agents.

- **Time-varying averaging matrices**. It means that each consensus step can potentially use a different matrix from the same spectral class $\mathcal{W}_{(-\lambda,\lambda)}$. In particular, since $\lambda < 1$, each averaging matrix is associated with a connected graph (see Proposition 2.12) As explained in Section 4.4, in that case, the conditions from Corollary 4.8 are applied separately to each consensus steps in PEP.

In both cases, the step-sizes of the algorithms are optimized for each value of λ. We also compare the algorithm performances with the following simple lower bound, valid for any algorithm in $\mathcal{A}_D$ that uses at most three consensus steps per iteration.

Proposition 8.6 (Lower complexity bound for $\mathcal{P}_x$)**.** *Let $A \in \mathcal{A}_D$ be any decentralized optimization method with updates using at most 3 consensus steps and information from past iterates only. There are local functions $f_i \in \mathcal{F}_{\mu,L}$ $(i = 1,\dots,n)$ and an averaging matrix in $W \in \mathcal{W}_{(-\lambda,\lambda)}$ such that the iterate error $\mathcal{P}_x(t) = \frac{1}{n}\sum_{i=1}^n \|x_i^t - x^*\|^2$, after t iterations of algorithm A using W, can be lower-bounded as*

$$\mathcal{P}_x(t) \geq \rho^t \mathcal{P}_x(0), \quad \text{with } \rho = \max\left\{\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^2, \lambda^3\right\}, \tag{8.25}$$

where $\kappa = \frac{L}{\mu}$ is the condition number of the local functions.

Proof. The lower bound (8.25) is obtained by combining the lower bound of centralized algorithms and the lower bound of a linear average consensus protocol. In the same settings, a decentralized optimization algorithm cannot converge faster than a centralized algorithm (in terms of number of iterations). For smooth and strongly convex functions, the centralized lower complexity rate for a first-order optimization algorithm that linearly combines information from past iterates is $\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^2$ [Nes18, Theorem 2.1.13]. Moreover, a decentralized optimization algorithm needs to achieve consensus between the agents. For communications, algorithm $A \in \mathcal{A}_D$ can only use linear average consensus steps (2.18) in its updates and can therefore not reach consensus faster than pure linear average consensus steps (2.7), when it applies to local functions with sufficiently different optimal solutions. The convergence rate of pure average consensus steps with averaging matrices from $\mathcal{W}_{(-\lambda,\lambda)}$ is λ, see Corollary 2.10. Since each iteration of algorithm A can perform at most 3 consensus steps, the best consensus convergence rate that can be achieved during one iteration of A is λ^3. □

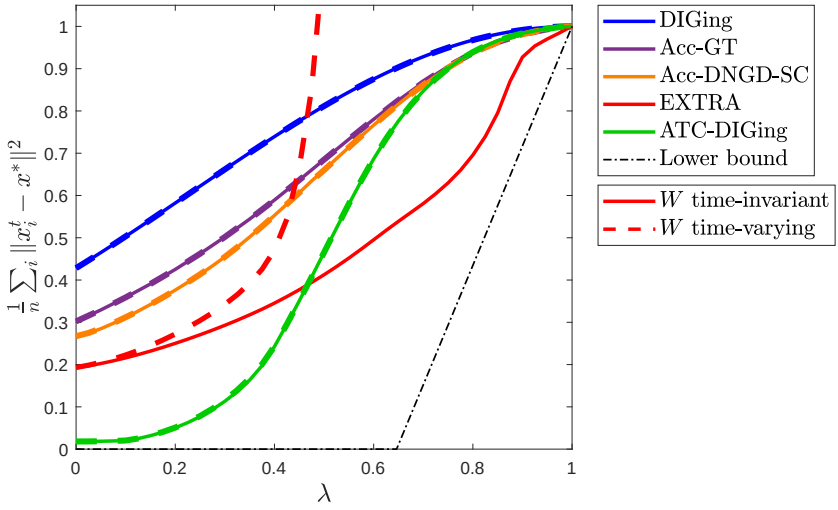


Fig. 8.12 Comparison of the PEP worst-case iterate error $\mathcal{P}_x$ (8.24) of different algorithms, after $t = 10$ iterations with time-invariant and time-varying averaging matrices. Evolution is shown with the range of eigenvalues $[-\lambda, \lambda]$. Local functions $f_i \in \mathcal{F}_{\mu,L}$ for all i ($\mu = 0.1$, $L = 1$) and optimized constant step-sizes.

8.5.1 Analysis of the results

Different analyses and observations can be made in Fig. 8.12.

Robustness to time-varying averaging matrices We see in Fig. 8.12 that, unlike EXTRA, the DIGing, ATC-DIGing, Acc-GT, and Acc-DNGD-SC algorithms have exactly the same worst-case performance for time-varying or time-invariant matrices and are therefore robust to time-varying connected networks. All these methods rely on the gradient tracking technique, which uses a dynamic average consensus protocol to allow agents to estimate the value of the global gradient. This suggests that the gradient tracking technique is effective for handling time-varying averaging matrices. This was known for DIGing and Acc-GT but not for ATC-DIGing

and Acc-DNGD-SC, for which we provide the first[6] worst-case analysis with time-varying graphs. Moreover, the worst-case behavior of EXTRA with time-varying matrices was also unknown since its theoretical analyses, e.g., [SLWY14, LL20], only focus on time-invariant settings. As we can see in Fig. 8.12, EXTRA is not robust to time-varying averaging matrices since its performance (after $t = 10$ iterations) with time-varying averaging matrices is always larger than with time-invariant ones, except when λ is close to 0. When λ increases, the EXTRA performance with time-varying averaging matrices explodes. This observation is consistent with the fact that the Lyapunov analysis from [SVSL20] (see Section 3.5) does not provide any convergence guarantee for EXTRA with similar settings when $\lambda \geq 0.6$. Indeed, this methodology evaluates the performance over one iteration of the algorithm and can therefore only consider the case where the averaging matrices are time-varying.

Accelerated algorithms While Acc-GT has a better proven convergence rate (for the functional error) than Acc-DNGD-SC, Fig. 8.12 shows that both algorithms perform similarly after $t = 10$ iterations for criterion $\mathcal{P}_x$ (8.24). Acc-DNGD-SC is even a bit better than Acc-GT for small λ (up to 13%). Moreover, while the algorithms are accelerated and have (proven) convergence rates with a better dependence on the function conditioning, they do not perform better than ATC-DIGing or EXTRA (with time-invariant matrices) after $t = 10$ iterations. Similar observations hold for $t = 20$ and $t = 30$. We expect that the convergence rates of Acc-GT and Acc-DNGD-SC have a large hidden constant and should become effective after a larger number of iterations, or for functions with worse conditioning (i.e., smaller μ).

Best performance after $t = 10$ iterations For time-varying averaging matrices, ATC-DIGing provides the best worst-case performance after $t = 10$ iterations, among the tested algorithms (see Fig. 8.12). For time-invariant averaging matrices, it still provides the best performance for $\lambda \leq 0.45$, but for larger λ, it becomes EXTRA that has the best worst-case performance after $t = 10$ iterations. In any case, the algorithms do not reach the lower bound (8.25).

[6] Only numerical simulations, with specific given functions, have been performed over time-varying graphs in [NOSU17, QL19].

8.5.2 Adapt-then-Combine (ATC) variants

In Fig. 8.12, we observe that the *adapt-then-combine* (ATC) variant of DIGing greatly improves its performance after $t = 10$ iterations. In Fig. 8.13 below, we have also analyzed the convergence rate of both algorithms with PEP, via the results of Proposition 8.2 which holds both for DIGing and ATC-DIGing. The figure shows convergence rates valid for any step of the algorithms and confirms the improved performance of ATC-DIGing over DIGing, especially for small λ values. In our settings ($\mu = 0.1$ and $L = 1$), the convergence rate of ATC-DIGing improves up to 26% over that of DIGing. This means that every 8 iterations, the ATC-DIGing error improves by a factor of 10 compared to that of DIGing. As for the rest of the section, the performance is computed for $n = 2$ agents but is valid for any $n \geq 2$, by Corollary 6.14.

The two algorithms are almost identical but differ in the order of their actions. In DIGing, the agents communicate and combine their variables x_i^k and s_i^k ($i = 1, \dots, n$), then they use the result to adapt their variables using their own local variables and functions. This is a *combine-then-adapt* (CTA) scheme.

(CAT)-DIGing (for $k = 1, \dots,$ and $i = 1, \dots, n$)

$$\text{Combine: } y_i^k = \sum_{j=1}^{n} w_{ij}^k x_j^k, \quad \text{then, Adapt: } x_i^{k+1} = y_i^k - \alpha s_i^k,$$

$$\text{Combine: } z_i^k = \sum_{j=1}^{n} w_{ij}^k s_j^k, \quad \text{then, Adapt: } s_i^{k+1} = z_i^k + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k).$$

In ATC-DIGing, the agents first adapt their variables using their own local variables and functions, and then they communicate the result and combine what they receive from their neighbors. The second *adapt-then-combine* update depends on the result of the first one.

ATC-DIGing (for $k = 1, \dots,$ and $i = 1, \dots, n$)

$$\text{Adapt: } y_i^k = x_i^k - \alpha s_i^k, \qquad \text{then, Combine: } x_i^{k+1} = \sum_{j=1}^{n} w_{ij}^k y_j^k,$$

$$\text{Adapt: } z_i^k = s_i^k + \nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k), \qquad \text{then, Combine: } s_i^{k+1} = \sum_{j=1}^{n} w_{ij}^k z_i^k.$$

While, in DIGing, both communications involve information from the previous iterate (x_i^k and s_i^k), ATC-DIGing includes information from the newest iterate (x_i^{k+1}) in the second communication. Therefore, unlike DIGing, both communicated variables cannot be shared at the same time. However, the number of variables communicated is exactly the same. ATC-DIGing uses the most updated information, which can explain its performance improvement over DIGing.

The two algorithms have already been compared with coordinated constant step-size in the appendix of [NOSU17]. Even if the analysis led to similar conclusions, it relied on conservative convergence rate guarantees, which could have led to a different analysis. The rates in [NOSU17] are of the same nature as the guarantees for DIGing discussed in Section 8.2, and in particular, they depend on the system size n. However, by Theorem 6.13, we know that the worst-case performance of DIGing and ATC-DIGing with settings described in this section (including coordinated step-sizes) are independent of the number of agents n.

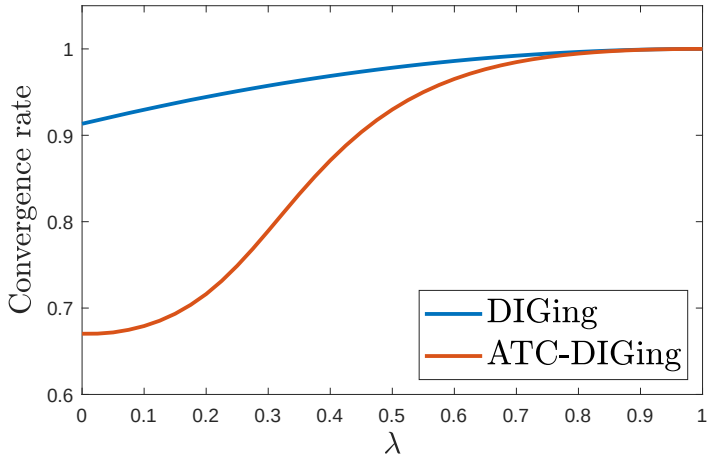


Fig. 8.13 Comparison of the convergence rates of DIGing and ATC-DIGing, as a function of the second-largest eigenvalue modulus λ and with time-varying averaging matrices. The rates are computed via PEP thanks to Proposition 8.2 with coefficient $\gamma = \frac{\alpha}{L}$. Computations are done for $n = 2$, but by Corollary 6.14, the results are valid for any $n \geq 2$.

Other algorithms with ATC variant The ATC scheme can be used in other algorithms, for example in Acc-DNGD-SC and Acc-GT. First experiments show that it makes Acc-DNGD-SC and Acc-GT less sensitive to the step-size choice and improves their performance after $t = 10$ iterations, as it is also the case for ATC-DIGing. Since the averaging matrices are columns stochastic, i.e., $\mathbf{1}^T W = \mathbf{1}^T$, the ATC-variant does not modify the agent average iterates. Hence, one could directly extend the agent average performance analysis of Acc-DNGD-SC and Acc-GT to their ATC variant. We do not know if ATC variants improve the asymptotic convergence rate of the algorithms, but our preliminary experiments show that they at least improve their hidden constant, leading to better worst-case performance in practice. Therefore, we recommend adopting the ATC scheme for gradient tracking based algorithms, in order to have efficient practical decentralized optimization algorithms.

In any domain,
Communication is key,
For things to succeed.

9

On the Optimal Communication Weights

We seek to identify the best communication weights to use in decentralized optimization algorithms, by leveraging the Performance Estimation Problem framework (PEP), that we adapted to decentralized optimization in Part I of this thesis.

In general, the performance of a decentralized optimization method is largely impacted by the choice of the mixing matrix W, see for example the survey [NOR18]. While the zero elements are imposed by the topology of the given graph $\mathcal{G}$, the values of the non-zero elements can be freely chosen and could be carefully determined to obtain efficient algorithms, as for any other parameter of a method. We restrict our search to mixing matrices that correspond to valid averaging matrices (see Definition 2.7).

Almost all theoretical performance guarantees from the literature of decentralized optimization algorithms with undirected graphs depend on the Second-Largest Eigenvalue[1] Modulus (SLEM) of W (after one), denoted λ (2.8), or some equivalent measure, because it is easy to use in the proofs and characterizes well the matrix behavior in terms of consensus (see Corollary 2.10).

[1] For directed graphs, the performance guarantees generally rely on the second-largest singular value of W.

We recall that the SLEM of an averaging matrix W is computed as

$$\lambda = \left\| W - \frac{\mathbf{1}\mathbf{1}^T}{n} \right\|_2 = \max\{\, |\lambda_2(W)|,\ |\lambda_n(W)| \,\},$$

where the real eigenvalues of W are labelled in descending order:

$$1 = \lambda_1(W) > \lambda_2(W) \geq \cdots \geq \lambda_n(W).$$

We assume that $\lambda < 1$, meaning the underlying graph is connected (see Proposition 2.12), otherwise, the agents never converge to a common point, see Corollary 2.10.

In general, theoretical worst-case performance guarantees improve as λ decreases. Therefore, the classical theoretical approach for selecting the weights in distributed optimization is to choose those leading to the smallest possible λ for the given network topology, which results in the smallest error guarantee. Moreover, choosing the weights minimizing λ is known to be optimal for pure linear consensus protocol, see Corollary 2.10 and [XB04]. However, this chapter shows that this prevalent strategy of minimizing the SLEM of the averaging matrix for selecting communication weights is not optimal for distributed optimization algorithms.

Contributions In this chapter, we formulate an optimization problem that computes the optimal communication weights for a given distributed optimization algorithm, a given undirected network topology, and other given settings. The formulation relies on the exact agent-dependent PEP formulation, developed in Chapter 4, which allows numerically computing the exact worst-case performance of an optimization algorithm with a given averaging matrix. The resulting weight-tuning problem has no guarantee to be smooth or convex, but we can obtain good estimates of the solution via a zero-order method. We show that in many settings, the optimal weights are different from those minimizing λ and allow, for example, to decrease the convergence time by up to 4 for the DIGing algorithm [NOS16]. It seems that all the eigenvalues, as well as their sign, are important to determine the performance of a decentralized optimization method. This indicates that the second-largest eigenvalue modulus λ (SLEM) is not the best determinant of weighted network performance in distributed optimization methods. We also explore different weights heuristics and compare them with the optimal ones, to find a better characterization of network performance in distributed optimization methods.

This chapter is based on the following paper [CH24b].

9.1 Weights Heuristics

We review possible heuristics to set the communication weights in distributed optimization, for which we will analyze the resulting performance in Section 9.4.

Let $B \in \mathbb{R}^{n\times m}$ be the oriented incidence matrix of the undirected graph $\mathcal{G}(\mathcal{E}, \mathcal{V})$, defined, for each node $i = 1, \dots, n$ and edge $e = 1, \dots, m$, as

$$B_{ie} = \begin{cases} 1 & \text{if edge } e \text{ starts from node } i, \\ -1 & \text{if edge } e \text{ ends at node } i, \\ 0 & \text{otherwise.} \end{cases}$$

Since the edges are undirected, any orientation can be chosen for each edge. The matrix B allows defining a valid averaging matrix W, i.e., a matrix satisfying all the assumptions from Definition 2.7, for any choice of the edge weights:

$$W = I - B\text{diag}(w)B^T, \tag{9.1}$$

where $w \in \mathbb{R}^m$ is the vector of weights for the m edges in $\mathcal{E}$ and $\text{diag}(w) \in \mathbb{R}^{m\times m}$ is a diagonal matrix with w on the diagonal.

9.1.1 Conventional choices of weights from average consensus

Common choices for the averaging matrix W in decentralized optimization are those developed for the pure linear averaging consensus protocol,

$$\mathbf{x}^{k+1} = (W \otimes I_d)\mathbf{x}^k, \tag{9.2}$$

and are listed below.

Minimum-λ weights. In [XB04], the authors have proved that the per-step convergence factor for the averaging consensus (9.2) is given by $\lambda = \|W - \mathbf{1}\mathbf{1}^T/n\|_2$, and therefore, that the fastest averaging matrix for the consensus, denoted W_{λ^*}, is obtained by minimizing λ,

$$W_{\lambda^*} = I - B\text{diag}(w_{\lambda^*})B^T, \tag{9.3}$$
$$\text{where } w_{\lambda^*} = \underset{w\in\mathbb{R}^m}{\text{argmin}} \left\| I - B\text{diag}(w)B^T - \frac{\mathbf{1}\mathbf{1}^T}{n} \right\|_2.$$

While computing W_{λ^*} requires global knowledge of the network, some papers have explored ways to compute it in a decentralized way, e.g., [BGPS06].

Uniform edge weights. When we choose all the weights to be equal to $q \in \mathbb{R}$, then we have $w = q\mathbf{1}$ and

$$W_q = I - qBB^T.$$

According to [XB04], the value of q minimizing the consensus convergence factor λ, is given by

$$q_{\lambda^*} = \frac{2}{\lambda_2(L) + \lambda_n(L)}, \tag{9.4}$$

where $L = BB^T$ is the Laplacian matrix of the graph $\mathcal{G}$ and its eigenvalues $\lambda_i(L)$ are labelled in ascending order. When the network G is connected, the resulting averaging matrix guarantees the convergence of the averaging consensus (9.2), i.e., $\lambda < 1$.

Maximum-degree weights. Another uniform edge weight that always yields asymptotic average consensus is

$$q_d = 1/(d_{\max} + 1), \tag{9.5}$$

where $d_{\max}$ is the maximal degree in $\mathcal{G}$, which can easily be computed in a decentralized manner.

Metropolis weights. The Metropolis weights only use local-degree information so that each node can set up its weights without knowing global information on the network. They are defined, for each edge $e = (i, j) \in E$, as

$$w_e = \frac{1}{\max\{d_i, d_j\} + 1},$$

where d_i is the degree of node i, without counting the node itself. The averaging matrix calculated from (9.1) with Metropolis weights is denoted W_M and guarantees the convergence of the averaging consensus (9.2). These weights are derived from the Metropolis–Hastings algorithm, when employed for the simulation of a Markov chain with uniform equilibrium distribution, see [BDX04].

Lazy Metropolis weights. A popular variation of Metropolis weights, introduced by [Ols15, Ols17], is defined as

$$W_{\text{lazy}-M} = \frac{1}{2}(I + W_M).$$

This averaging matrix is diagonally dominant and has thus non-negative eigenvalues, which can be useful for proofs.

9.1.2 Other weight heuristics based on the full spectrum

In the pure consensus (9.2), the decrease of the consensus error $\|\mathbf{x}^k - \mathbf{1} \otimes \overline{x}\|$ is dominated by λ because the influence of all the other smaller eigenvalues (in absolute value) vanishes more rapidly, and the influence of $\lambda_1 = 1$ is zero because the error vector is orthogonal to the associated eigenvector $\mathbf{1}$. In decentralized optimization, the consensus is perturbed by local gradient updates, which prevent the vanishing of smaller eigenvalues of the averaging matrix. Therefore, all the eigenvalues may influence the convergence speed of decentralized optimization algorithms. This motivates us to consider weight heuristics based on all the eigenvalues of the averaging matrices, and that we consider potentially relevant in decentralized optimization.

Minimum-$\|W\|_\Sigma$ weights. The nuclear norm of a symmetric matrix W is defined as

$$\|W\|_\Sigma = \sum_{i=1}^{n} |\lambda_i(W)|,$$

and the weighted nuclear norm[2] is given by

$$\|W\|_{\Sigma,\gamma} = \sum_{i=1}^{n} \gamma_i |\lambda_i(W)|,$$

where $\gamma = [\gamma_1, \ldots, \gamma_n]$, with $\gamma_i \geq 0$ a non-negative weight assigned to λ_i. Both norms are used as convex surrogates for the matrix rank in problems involving rank minimization. The weights minimizing $\|W\|_{\Sigma,\gamma}$ can be relevant in distributed optimization because it generalizes the weights minimizing λ, by taking into account all the other eigenvalues, which may play a role in the algorithm convergence. We denote the corresponding

[2] The weighted nuclear norm is not a true norm, as it does not always satisfy the triangle inequality.

averaging matrix as $W_{\Sigma^*,\gamma}$ and define it formally as

$$W_{\Sigma^*,\gamma} = I - B\text{diag}(w_{\Sigma^*,\gamma})B^T,$$
$$\text{where } w_{\Sigma^*,\gamma} = \underset{w\in\mathbb{R}^m}{\text{argmin}} \|I - B\text{diag}(w)B^T\|_{\Sigma,\gamma},$$

which can be computed by solving an equivalent SDP, described in [Ali95].

Minimum-R_{tot} weights. The total effective resistance of a weighted graph, denoted R_{tot}, can be defined based on the eigenvalues of the associated averaging matrix W:

$$R_{\text{tot}}(W) = n\sum_{i=2}^{n}\frac{1}{1-\lambda_i(W)}. \tag{9.6}$$

The initial interpretation and definition of R_{tot} is related to the total electrical resistance of a resistor network with conductances given by the edges. [GBS08] provide many useful interpretations of the total effective resistance. For example, R_{tot} is related to the average commute time in the Markov Chain derived from the weighted graph. Hence, the averaging matrix $W_{R^*_{\text{tot}}}$ minimizing the total effective resistance would consist in the graph leading to the smallest average commute time:

$$W_{R^*_{\text{tot}}} = I - B\text{diag}(w_{R^*_{\text{tot}}})B^T, \tag{9.7}$$
$$\text{where } w_{R^*_{\text{tot}}} = \underset{w\in\mathbb{R}^m}{\text{argmin}}\ R_{\text{tot}}(I - B\text{diag}(w)B^T)$$
$$\text{s.t.} \sum_{e \text{ adjacent to } i} w_e \leq 1-\epsilon \quad \text{for all } i,$$

where $\epsilon \in [0,1)$ is the smallest accepted value for the diagonal elements of the averaging matrix. Using results from [GBS08], the solution to problem (9.7) can be computed by solving an equivalent SDP.

Least mean-square deviation weights. In [XBK07], the auhtors formulate a convex problem to compute the symmetric averaging weights that minimize the mean-square deviation from the mean produced by the average consensus with additive noise. This extension of the averaging iteration (9.2) adds noise at each node and each step:

$$\mathbf{x}^{k+1} = (W \otimes I_d)\mathbf{x}^k + \mathbf{v}^k, \tag{9.8}$$

where $\mathbf{v}^k = [v_1^k \dots v_n^k]^T$, and $v_i^k \in \mathbb{R}^d$ are independent random variables, identically distributed, with zero mean and unit variance. Due to the noise, the agent values x_i are not converging to the average, but the quality of the consensus can be evaluated using the mean-square deviation from the average $\overline{x} = \frac{1}{n}\sum_i x_i$:

$$\delta^k = \mathbb{E}\sum_{i=1}^{n}(x_i^k - \overline{x})^2$$

According to [XBK07], δ^k converges to a steady state mean-square deviation which depends on the eigenvalues of W as

$$\delta_{ss}(W) = \lim_{k\to\infty} \delta^k = \sum_{i=2}^{n} \frac{1}{1-\lambda_i(W)^2}.$$

The averaging matrix W_{δ^*} minimizing the steady state mean-square deviation is therefore given by

$$\begin{aligned} W_{\delta^*} &= I - B\mathrm{diag}(w_{\delta^*})B^T, \\ \text{where } w_{\delta^*} &= \operatorname*{argmin}_{w\in\mathbb{R}^m} \delta_{ss}(I - B\mathrm{diag}(w)B^T) \end{aligned} \tag{9.9}$$

The authors of [XBK07] show that problem (9.9) is convex and details how to evaluate the function δ_{ss}, as well as its gradient and hessian, such that it can be minimized using standard optimization methods. Choosing W_{δ^*} for distributed optimization may be relevant since, distributed optimization algorithms are similar to the noisy consensus (9.8), where additive noise is replaced by a local update based on local gradients information, that may not be i.i.d., of zero mean or unit variance.

9.2 Worst-case Performance evaluation

To evaluate the quality of an averaging matrix W for a given decentralized algorithm, we compute the worst-case performance it provides, via the Performance Estimation Problem framework (PEP). In particular, we rely on the exact agent-dependent PEP formulation, tailored for decentralized optimization and developed in Chapter 4.

As detailed in Chapter 4, to obtain a tight bound on the performance of an algorithm A, the conceptual idea of PEP is to search for instances

of local functions and starting points for all agents, allowed by the setting considered, producing the largest error after a given number t of iterations of the algorithm. This idea can be formulated as a real optimization problem that maximizes the error measure $\mathcal{P}$ of the algorithm result, over all possible functions and initial points allowed, see PEP Problem (4.1). The worst-case performance $w(\mathcal{S}_n)$ computed by Problem (4.1) depends on the performance evaluation settings $\mathcal{S}_n$ (4.2). In this chapter, we define it as a function of the step-size parameter of the algorithm α and the averaging matrix W:

$$E_{\mathcal{S}_n}(W,\alpha) = w\left(\mathcal{S}_n = \{n, A(\alpha), t, \mathcal{P}, \mathcal{F}, \mathcal{I}, \{W\}\}\right). \tag{9.10}$$

The worst-case performance function $E_{\mathcal{S}_n}(W,\alpha)$ is always specific to a given setting $\mathcal{S}_n$, but it will sometimes be ignored in the notation $E(W,\alpha)$.

The PEP framework (4.1) allows evaluating the worst-case performance function $E(W,\alpha)$ for the different weight heuristics from Section 9.1 on different graphs and algorithms, and to compare their resulting worst-case performance. Furthermore, it also allows for seeking the weights leading to the best performance, by minimizing $E_{\mathcal{S}_n}(W,\alpha)$.

9.3 Weights minimizing the worst-case performance

To find the best averaging matrix to use for a setting $\mathcal{S}_n$ and a graph $\mathcal{G}$, we can find the averaging W^*, along with a step-size α^*, minimizing the performance function $E_{\mathcal{S}_n}(W,\alpha)$ of the algorithm, defined in (9.10) and computed via PEP (4.1).

$$\begin{aligned}(W^*,\alpha^*) = \underset{\substack{W\in\mathbb{R}^{n\times n}\\ \alpha\geq 0}}{\operatorname{argmin}}\ & E(W,\alpha) \\ \text{s.t. } & W = W^T, \quad W\mathbf{1} = \mathbf{1}, \quad W \in \mathcal{T}. \\ & \lambda(W) = \|W - \mathbf{1}\mathbf{1}^T/n\|_2 < 1.\end{aligned} \tag{9.11}$$

The three constraints on the first row correspond to the definition of a (symmetric) averaging matrix, see Definition 2.7, with $\mathcal{T}$ corresponding to the set of matrices matching the topology of the given graph $\mathcal{G}$. The constraint $\lambda(W) < 1$ is necessary for the convergence of decentralized algorithms (see Corollary 2.10). Matrices that do not satisfy it should, in

principle, lead to larger values of E, when the number t of iterations is large enough. To ensure the condition is met, even for small t, we interpret E as ∞ when it does not hold. Therefore, Problem (9.11) can be written without constraints, using (9.1),

$$(w^*, \alpha^*) = \underset{\substack{w \in \mathbb{R}^m \\ \alpha \geq 0}}{\operatorname{argmin}} E(I - B\operatorname{diag}(w)B^T, \alpha), \tag{9.12}$$

with step-size $\alpha \in \mathbb{R}$ and edge weights vector $w \in \mathbb{R}^m$ as variables, and where we interpret $E(W, \alpha)$ as ∞ whenever $\|W - \mathbf{1}\mathbf{1}^T/n\|_2 \geq 1$. This problem is a priori non-convex and non-smooth. While this is difficult to prove, it could be smooth or convex for some settings. A simple way of solving Problem (9.12) is using a zero-order method, that only relies on function evaluations to identify a minimizer. In this work, we use the pattern search method from Matlab (also known as direct search), which is described and analyzed in [ADJ02]. Their analysis showed that even if the objective function is discontinuous or extended-valued, the method finds a limit point with some minimizing properties. Moreover, if the function is strictly differentiable at this limit point, then it corresponds to a local minimum. We have, of course, no guarantee to converge to a global minimizer in the non-convex case. There are other possible approaches to solve (9.12). For example, one could try to solve Problem (9.12) using first-order methods, which would allow converging to a local minimum. The generalized gradient of function $E(W, \alpha)$ can be computed via an SDP sensitivity analysis, see [BS13]. Indeed, the PEP problem (4.1) to solve for computing $E(W, \alpha)$ can be reformulated as an SDP, as explained in Section 4.2.

For this analysis, we have decided to use the simple *pattern search* zero-order method [ADJ02], which already provides interesting results, as shown in the following section.

9.4 Results

Using the *pattern search* method from Matlab, we solve problem (9.12) for different performance settings $\mathcal{S}_n$ and show the results in Fig. 9.1. The figure compares the optimal weights with the heuristics weights presented in Section 9.1. To obtain a fair comparison, we tune the step-size α of the methods, for each heuristic. We analyze averaging matrices for the follow-

ing algorithms, with a constant (optimized) step-size α,

- DIGing [NOS16], see Algorithm 3;
- ATC-DIGing [NOSU17], see equations (2.14)-(2.15);
- EXTRA [SLWY14], see Algorithm 2 with $\tilde{W} = \frac{I+W}{2}$;
- Acc-DNGD-SC [QL19], which is a variant of Algorithm 4 with constant parameters, and that is suited to smooth and strongly convex functions.

We have decided to analyze these algorithms because of their significance in the literature. As explained in Section 8.2.3, for DIGing and ATC-DIGing, we can compute a bound on the linear convergence rate by analyzing only $t = 1$ iteration of the algorithm, with a performance criterion (8.10) identical to the initial condition. For the other algorithms (EXTRA and Acc-DNGD), we compute the decrease after $t = 5$ steps. We assume local functions to be L-smooth and μ-strongly convex (see Definition 3.2), so that all the above algorithms exhibit a linear convergence. We consider 4 usual network topologies for n nodes: complete, star, cycle, and grid. Details of these topologies are given in the table header from Fig. 9.1. We mainly focus on the situation with $n = 9$ nodes. This small number of nodes gives us a better intuition of the results and also avoids excessive computational load to solve the tuning problem (9.12). To further reduce this computational load, we have decided to give equal weights to equivalent edges (see Definition 9.1 below), so that the actual number of variables in the problem is very low. This corresponds to a restriction on the searching space of (9.12), which may lead to suboptimal solutions if the problem is not convex in W.

Definition 9.1 (Equivalent edges, [GYA18])**.** Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be a graph and $e_1, e_2 \in \mathcal{E}$ two edges of the graph. If there is an automorphism mapping e_1 to e_2, then there is an equivalence relation between e_1 and e_2. The equivalence classes of edges of $\mathcal{G}$ are called the edge orbits.

For example, the complete graph only has one orbit with all the edges. This is also the case for the star graph and the cycle graph. Therefore, we only have one constant edge weight to optimize for these topologies. The grid graph with 9 nodes, shown in Fig. 9.1, has 2 edge orbits: one with the outer edges and the other with the inner edge. For larger grid graphs, each of these orbits divides in two so that we have 4 edge orbits. We may expect better optimal values for the tuning problem (9.12) when allowing all the

weights to be different, however, we have observed on small networks ($n = 3, 4, 5$), that this is not the case for the algorithms considered and that we still obtain equal weights for equivalent edges, even when not imposed in advanced. Moreover, all the weights heuristics presented in Section 9.1 satisfy this equivalent weight property.

9.4.1 Weights minimizing λ (SLEM) are not optimal

As shown in Fig. 9.1, the averaging matrix W_{λ^*} (9.3), minimizing λ (SLEM) for the given graph $\mathcal{G}$, does not provide the best convergence guarantees for the tested decentralized algorithms, even with a step-size α specifically optimized for these weights. This observation is counter-intuitive because small λ values correspond to well-connected graphs and because W_{λ^*} is optimal for pure consensus steps (9.2). All known theoretical performance bounds in decentralized optimization depend on λ, and improve when λ decreases, however, our results show that for a given topology, minimizing λ does in general not optimize worst-case performance. This suggests that λ may not be the best characterization of the network performance in distributed optimization methods. Even for the complete graph, where a reasonable person would guess that $W_{\lambda^*} = \frac{\mathbf{1}\mathbf{1}^T}{n}$ would be optimal, we can obtain improvement on the performance by choosing W^* with a different unique eigenvalue.

In the settings of Fig. 9.1, choosing the optimal averaging matrix W^* (in green), instead of W_{λ^*} (in red) improves the iterates convergence rate ρ of DIGing and ATC-DIGing by up to 5% and their corresponding convergence time $\tau = 1/\log(1/\rho)$ is up to 4 times smaller. For EXTRA and Acc-DNGD, their worst-case performance after 5 iterations is improved by up to 18%. To have a fair comparison, for each choice of averaging matrix, Fig. 9.1 shows its corresponding performance, for the best possible step-size. We have observed that the optimal weights W^* allow the algorithms to work better with larger step-sizes α, which may partially explain their improved performance.

We observe that the sign of the eigenvalues of the averaging matrix and their distribution are related to the resulting performance. Indeed, in many cases (DIGing, ATC-DIGing, Acc-DNGD), the optimal averaging matrices have a smaller range of eigenvalues $|\lambda_2 - \lambda_n|$, with the leftmost (negative) eigenvalue λ_n significantly larger than that of W_{λ^*}, while the rightmost (positive) eigenvalue λ_2 is only slightly larger, which can barely be seen on the plots from Fig. 9.1. Matrix W_{λ^*} always have $\lambda_n = -\lambda_2$ but by taking

advantage of non-symmetric range of eigenvalues, W^* reach smaller $|\lambda_2 - \lambda_n|$. For EXTRA, things are surprisingly different, the optimal averaging matrix W^* has a larger range of eigenvalues, the smallest of which is close to -1. This should be investigated to see if the phenomenon persists for larger numbers t of iterations.

9.4.2 Weights heuristics analysis

As shown in Fig. 9.1, none of the heuristics from Section 9.1 achieves optimal performance. However, some of them come close in certain contexts, while others never seem to perform well. For example, the lazy-Metropolis weights $W_{\text{lazy}-M}$ (in black) do not perform very well in the settings we tested, because they give averaging matrices with too large positive eigenvalues. Its eigenvalues are concentrated in a smaller range but always have the largest (positive) eigenvalue among all the tested matrices. The standard Metropolis weights W_M present better results, while it can still be far from the optimal weights in some cases. Surprisingly, W_M often beats W_{λ^*}, especially for the grid topology.

For clarity, Fig. 9.1 omits the performance of the two heuristics for uniform edge weights (9.4) and (9.5). Their respective performance is always worst or equal to those of W_{λ^*} and W_M.

We observe that the averaging matrices that perform similarly tend to have the same eigenvalue distribution and have actually weights close to each other, even if similar[3] matrices are not guaranteed to be equal. Moreover, when the performance of a heuristic weight approaches the optimal performance, it also approaches the optimal weight values. This would be consistent with the convexity of problem (9.11) and the uniqueness of its optimal solution, which cannot be guaranteed theoretically at this stage.

No heuristic strictly outperforms the others in all situations. For DIGing, ATC-DIGing, and Acc-DNGD, the best heuristic seems to be W_{δ^*}, which performs very well, except for the star graph. By definition (9.9), W_{δ^*} minimizes the steady-state mean square deviation δ_{ss} of a consensus with additive noise. In distributed optimization, noise is replaced by all kinds of local updates, which take local gradients into account. In terms of eigenvalues, W_{δ^*} minimize $\delta_{ss} = \sum_{i=2}^{n} \frac{1}{1-\lambda_i(W)^2}$, which drives the matrix to have small eigenvalues in absolute value and penalizes more the large

[3]Similar, in the sense that the matrices have the same spectrum.

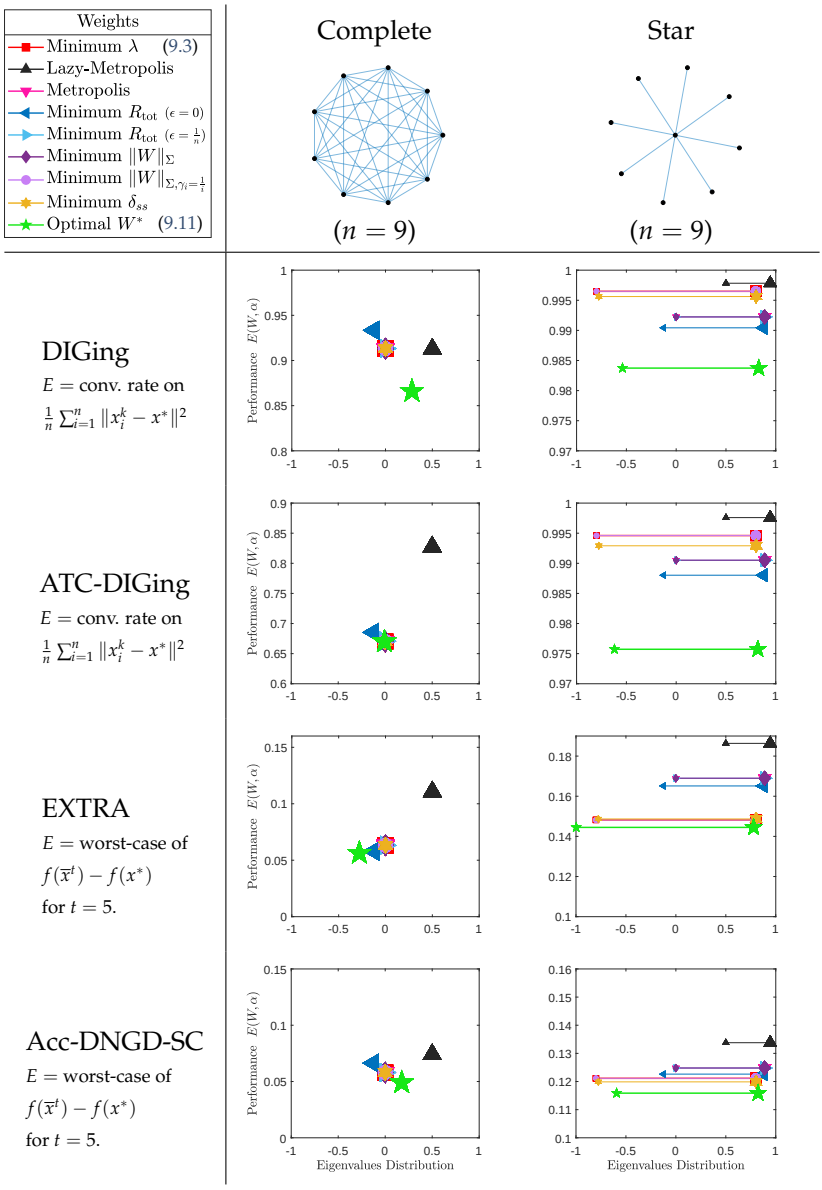


Fig. 9.1 These plots show the error criterion $E(W, \alpha)$ on the vertical axis, for the optimal averaging matrix W^* in comparison with different averaging matrix heuristics from Section 9.1. The plots also show the eigenvalue distribution of the matrices (except $\lambda_1 = 1$) on the horizontal axis. Each marker corresponds to a different eigenvalue, with a size proportional to its multiplicity. *The figure continues on the next page.*

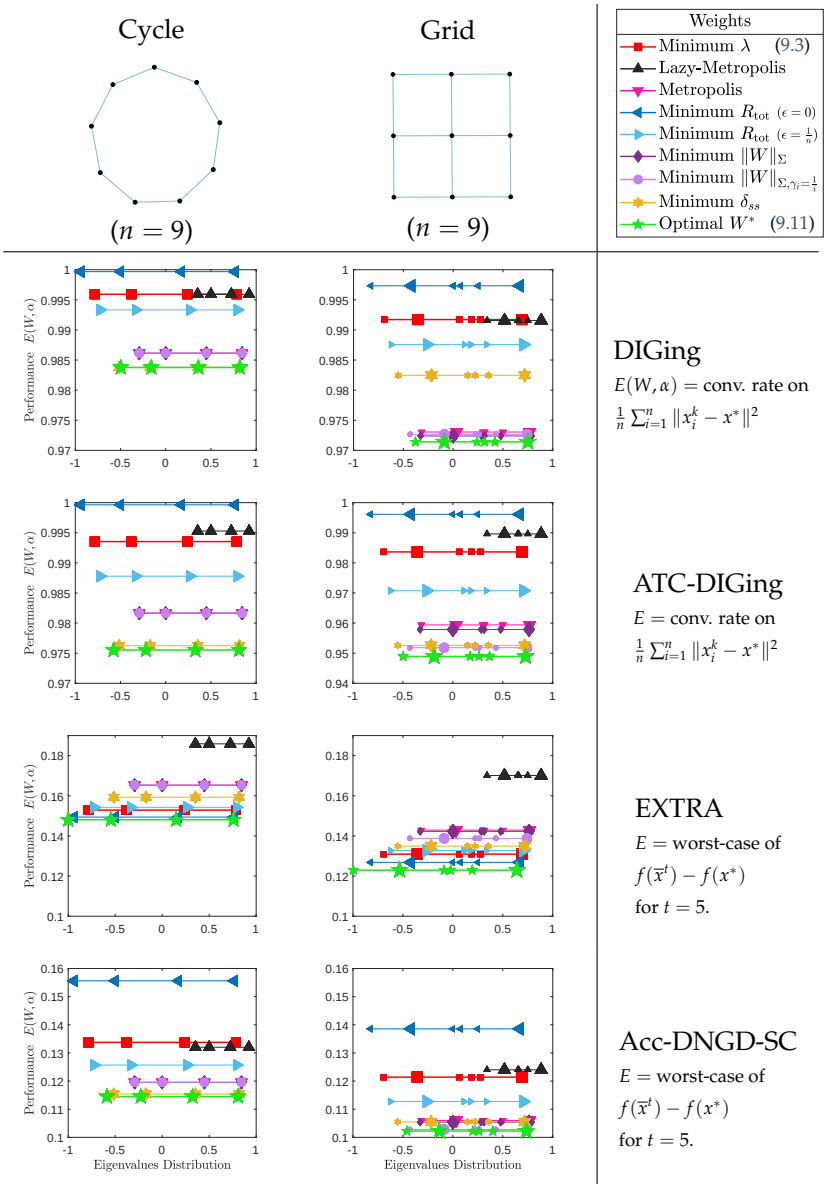


Fig. 9.1 *(continued)* To obtain a fair comparison between the averaging matrices, we tune the step-size α of the methods for each of them. Each plot corresponds to a different topology or a different algorithm. The local functions are μ-strongly convex and L smooth, with $\mu = 0.1$ and $L = 1$.

eigenvalues due to the square. This explains why the eigenvalues of W_{δ^*} often lie in a smaller range around zero than other matrices.

For EXTRA, the best heuristic seems to be $W_{R^*_{\text{tot}}}$, which selects the weights leading to the smallest (negative) eigenvalues to minimize R_{tot}, defined in (9.6). This strategy does not appear to be good for other algorithms.

9.4.3 Optimal weights

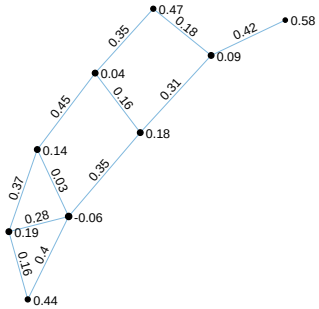
Unlike the weights heuristics, the optimal weights can be different depending on the algorithm. In Fig. 9.1, we can see that EXTRA always has optimal weights with very different eigenvalues distributions from other algorithms. DIGing, ATC-DIGing, and Acc-DNGD have more similar eigenvalue distributions, but their optimal weights are not exactly the same. This means that there is no weight heuristic or characterization that can be optimal for all decentralized algorithms from class $\mathcal{A}_D$.

9.4.4 Results for two random networks

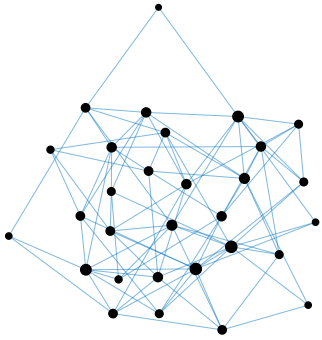
To complete our analysis, we look at two random graphs, sampled from the Erdos-Renyi $G(n,p)$ model and shown in Fig. 9.2. The convergence rates and times of DIGing on these graphs with optimal and heuristics weights are shown in Table 9.1. Here too, we observe the improvement of W^* over W_{λ^*}, leading to a convergence time more than 4 times shorter. The best heuristic also seems to be the least-mean square deviation averaging matrix W_{δ^*}, which achieves a performance close to the optimum.

Graph		W^*	W_{λ^*}	W_M	$W_{\text{lazy}-M}$	W_{δ^*}	$W_{R^*_{\text{tot}}}$	$W_{R^*_{\text{tot}},\epsilon}$	W_Σ	$W_{\Sigma,\gamma}$
Fig. 9.2a	ρ	0.9871	0.9971	0.9932	0.9982	0.9882	1.0735	0.9932	0.9932	0.9924
	τ	77	343	146	554	84	∞	146	147	131
Fig. 9.2b	ρ	0.9714	0.9874	0.9805	0.9942	0.9805	0.9930	0.9895	0.9835	0.9805
	τ	34	80	51	171	51	142	95	60	51

Table 9.1 Convergence rates ρ and convergence times $\tau = 1/\log(1/\rho)$ of the DIGing algorithm with optimal and heuristic averaging matrices, for both graphs from Fig. 9.2.



(a) $n = 9, m = 12$.
Sampled from $G(9, p = 0.4)$.



(b) $n = 30, m = 91$.
Sampled from $G(30, p = 0.2)$

Fig. 9.2 Two random graphs generated using Erdos-Renyi $G(n, p)$ model. For (a), the labels show the optimal weights W^* for DIGing.

9.5 Summary and conclusion

We showed how our agent-dependent PEP formulation from Chapter 4 can be used to compute the optimal communication weights for a distributed optimization algorithm over an undirected network. Our analysis revealed that the weights minimizing the second-largest eigenvalue modulus (SLEM) of the averaging matrix are suboptimal. While the SLEM characterizes well the convergence rate of a pure consensus protocol, we showed that this is not the best determinant for the performance of a weight matrix in distributed optimization. Some other weight heuristics give better results, but the best characterization of the network performance for distributed optimization is still an open question and probably involves all the eigenvalues of the averaging matrix.

PART III
Conclusion

Journey meets its end,
Glance back, then ahead,
Where do we go now?

10

Research outcomes and perspective

THIS THESIS developed and exploited tools for automatic and accurate computation of worst-case performance of decentralized optimization algorithms. Our tools rely on the Performance Estimation Problem (PEP) and are particularly helpful in analyzing, understanding, tuning, and comparing decentralized algorithms. Such tasks appear to be complex to treat analytically. In fact, existing theoretical worst-case performance and convergence analysis of decentralized optimization algorithms often rely on specific settings and conservative conditions (especially for the step-size), leading to analytic worst-case guarantees that may be too loose or complex. For all these reasons, our new tools appear to be essential for better use and faster development of efficient decentralized optimization algorithms.

To conclude the thesis, we summarize the main research outcomes related to the development and exploitation of these PEP tools. Then, we overview some research perspectives and open questions.

10.1 Research outcomes

Worst-case performance of a decentralized optimization method

We have developed several formulations of the Performance Estimation Problem (PEP) for decentralized optimization, that have different characteristics and are adapted to different settings. All our formulations can be written as a semidefinite program (SDP) and hence can be solved efficiently. Their development has also revealed some fundamental properties of worst-case performance of decentralized optimization algorithms. We first describe these general results, before reviewing our different PEP formulations.

General results We established conditions under which the worst-case performance of a decentralized algorithm is independent of the number of agents (see Theorem 6.13). In particular, it requires that all the agents play an equivalent role in the algorithm and its performance evaluation. These conditions are met for many algorithms and common performance evaluation settings. Agent-independent performance guarantees enable analyses, insights, and parameter tuning that are valid for any system size (including $n \to \infty$). This result can greatly simplify the derivation of such agent-independent guarantees since it allows analyzing the case with only $n = 2$ agents to get performance guarantees that are valid for systems of any size $n \geq 2$.

Moreover, when all agents are equivalent, we also showed that the worst-case local iterates are rotations (or reflections) of each other and the worst-case local functions are identical up to a change of variables (see Proposition 6.19).

The exact agent-dependent PEP formulation This is an optimization problem that seeks to identify the local variables and local function of each agent that result in the worst performance of a given decentralized algorithm (see Chapter 4). The algorithm is assumed to use a given averaging matrix, which also determines the communication network topology. The resulting SDP PEP has a size growing with the number of agents and provides exact worst-case performance, which is specific to the given averaging matrix.

The spectral agent-dependent PEP formulation This formulation looks for worst-case guarantees valid for an entire spectral class of symmetric averaging matrices, characterized by their range of eigenvalues, to obtain more general guarantees that are comparable with those obtained in the literature (see Chapter 4). This PEP formulation is similar to the exact agent-dependent PEP formulation, except for the representation of consensus steps that are used in the decentralized algorithm. Indeed, the effect of consensus steps cannot be described explicitly since the class of allowed averaging matrices encompasses a multitude of network topologies and weight strategies. Therefore, we proposed necessary conditions, which can be embedded in an SDP PEP, to describe the effect of a set of consensus steps using the same averaging matrix. The conditions are not sufficient, but we characterized the class of steps allowed by our proposed relaxed description. We also proved that no sufficient conditions can be incorporated into the SDP PEP. Hence, the resulting spectral PEP formulation uses our new necessary conditions to represent consensus steps and provides upper bounds on the worst-case performance of the decentralized algorithm that are valid for any averaging matrix from the given spectral class. The tightness of the result can be verified a posteriori by recovering a valid worst-case averaging matrix, which has been done in all our experiments. When the worst-case performance is known to be agent-dependent, we recommend using this spectral agent-dependent PEP formulation with $n = 2$, because it provides accurate bounds whose tightness can be checked and is easy to implement, especially with the PESTO toolbox [THG17].

The agent-independent relaxed PEP formulation Unlike agent-dependent formulations, this relaxed PEP formulation has a size independent of the number of agents. It also considers a spectral class of averaging matrices. However, it relaxes the separability of the global objective function in the decentralized problem (see Chapter 5). Therefore, this PEP formulation gives an upper bound on the worst-case performance of the given decentralized algorithm. The tightness of its result can only be verified by comparing it to a known tight bound, such as those obtained with the spectral agent-dependent formulation, which we recommend using in practice. However, this relaxed formulation provides interesting research directions for understanding the conditions that enable an algorithm to be robust to heterogeneity in the conditioning of local functions.

The compact PEP formulation This formulation exploits the symmetries and equivalences that exist between the agents in the problem, to reformulate the spectral agent-dependent PEP formulation in a compact manner, i.e., with size that depends on the number of agent equivalence classes rather than the total number of agents (see Chapter 6). The use of this compact formulation is especially beneficial in settings where we expect the worst-case performance to depend on the number of agents, such as the performance of the worst agent or when different groups of agents use different (uncoordinated) step-sizes, function classes, or initial conditions. Indeed, the compact formulation enables efficient analysis of such worst-case performances over systems of any size n (including when $n \to \infty$).

Analysis, tuning, and comparison of decentralized algorithms

By exploiting our new PEP formulations (mostly the spectral agent-dependent one), we were able to analyze, tune, and compare several algorithms (see Chapter 8). In particular, we obtained tight worst-case guarantees for DGD, DIGing, and Acc-DNGD, that largely improve on the literature bounds and provide much better step-size tuning. For these algorithms, we also identified a general form for the worst-case averaging matrix. Moreover, our PEP analysis shed light on conjectures from [QL19] about the convergence of Acc-DNGD with smooth convex functions. Our experiments showed that with an appropriate step-size, the algorithm can indeed achieve a worst-case asymptotic convergence rate of $\mathcal{O}\left(\frac{1}{t^2}\right)$.

Our PEP experiments confirmed that in many cases, worst-case performance is independent of the number of agents n. This is not the case of the performance of the worst agent, that we analyzed for the EXTRA algorithm, which scales sublinearly with n and reaches a plateau when a part of the worst agents are excluded from the performance criterion. Additionally, our experiments indicated that the average performance of EXTRA depends only on the conditioning of the global function, making it robust to the conditioning heterogeneity of local functions. On the other hand, EXTRA is not robust to the use of time-varying averaging matrices, whereas DIGing, ATC-DIGing, and Acc-DNGD are. Finally, the comparison of the algorithms have demonstrated how a slight modification of a decentralized algorithm can significantly improve its performance, namely, ATC-DIGing outperforms DIGing. It appeared that the Adapt-Then-Combine (ATC) scheme can also be applied successfully to other algorithms.

Communication weights in decentralized optimization

We used the exact agent-dependent PEP formulation to compute optimal communication weights for a distributed optimization algorithm over a given undirected network (see Chapter 9). Our analysis revealed that choosing the weights minimizing the second-largest eigenvalue modulus of the averaging matrix, while optimal for existing theoretical performance bounds, is generally not optimal regarding the exact worst-case performance of the algorithms. Other heuristics yield better results but none of them achieves optimality.

10.2 Perspectives and open questions

We first detail research perspectives regarding the development of the PEP techniques for decentralized optimization. Next, we mention future research directions that rely on our decentralized PEP techniques. Finally, we give our opinion on the development directions of decentralized optimization and explain how the contributions of this thesis can help progress in these directions.

Further development of our PEP tools

Future development of PEP for decentralized optimization could focus on other classes of averaging matrices, other types of methods, or a better understanding of the agent-independent relaxed PEP formulation.

Other classes of averaging matrices and networks Our work focused on communications over undirected and connected networks. Natural extensions would be to consider other types of communication networks in our PEP formulations, including, for example, directed graphs or B-connected graph sequences. We can obtain practical PEP representation for consensus steps over balanced directed graphs. Indeed, if we consider the class of non-symmetric and generalized doubly stochastic mixing matrices, with a given bound on the second-largest singular value σ, then our PEP consensus description (Corollary 4.8) extends naturally by replacing the range of eigenvalues by $[-\sigma, \sigma]$. However, the class of column stochastic matrices, which is the most common assumption with (unbalanced) directed communication, would be more difficult as it is not directly related to spectral properties of the matrices.

Other types of methods Our work focused on decentralized optimization methods that linearly combine variables, gradient samples, and consensus steps. While this contains numerous methods, it excludes different classes of methods, that would benefit from computer-aided analysis:

- **Push-sum based methods**, generally used to handle directed communication.
- **Adaptive decentralized methods**, which use the current iterates, gradients, or functions values to adjust their step-sizes.
- **Stochastic decentralized methods**, using a stochastic gradient oracle.

Push-sum based and adaptive methods generally involve products or quotients of variables, that cannot be embedded in an SDP PEP formulation. Ad-hoc modifications, changes of variables, or relaxations could allow our decentralized PEP formulation to analyze such methods. Alternatively, one could also consider formulating and solving non-convex PEP, as recently done for PEP analysis of centralized adaptive methods [DGFST23] or algorithm design [DGVPR24]. Adaptive and stochastic methods are widely used in machine learning but their performance and convergence properties are still not fully understood. With stochastic methods, the difficulty lies in computing tight worst-case guarantees for the *expected* performance. One can always obtain a loose upper bound by considering a PEP that seeks the worst-case realization of the stochastic noise. Further work is needed to efficiently and accurately analyze the performance of stochastic methods with PEP, in both centralized and decentralized versions.

Role of function separability A promising future work is to study the tightness of the agent-independent relaxed PEP formulation from Chapter 5 to understand when the separability of the global objective function plays a role, or not, in the worst-case performance of decentralized optimization methods. In particular, this could provide insights into what makes an algorithm robust to heterogeneity in local function properties. This is an interesting feature for a decentralized algorithm, as it broadens the scope of its applications and guarantees.

Further exploitation of our PEP tools

The versatility of our PEP tools opens up many other possibilities for application in decentralized optimization. In particular, one can use it to analyze the worst-case performance of other algorithms or settings and rapidly get answers to many questions.

Tuning and testing variants It can help to tune the parameters of a method and test ideas of algorithm variants and improvements, as we did for the ATC scheme.

ATC vs CTA It would also be interesting to further analyze the ATC variants of existing methods, to better understand the improvement that can be expected from using an ATC scheme instead of a CTA scheme, and why.

Performance determinant of averaging matrices Although the second-largest eigenvalue modulus of the averaging matrix characterizes well the convergence rate of a pure consensus protocol, we know from Chapter 9 that this is not the most appropriate determinant of the matrix performance for distributed optimization. Therefore, an interesting open question is to find a better matrix characterization to determine its performance in distributed optimization. As explained in Chapter 9, a good characterization should probably take into account all the eigenvalues of the averaging matrix. Moreover, a tight characterization of optimal weights can vary depending on the type of algorithm used.

Design of decentralized algorithms As detailed in Section 3.4, PEP has already been used to design optimal first-order centralized optimization methods in different settings. These techniques search for the algorithms minimizing the worst-case performance, which can be computed with PEP. Using the same approach, and our new decentralized PEP formulation, it could be possible to design optimal decentralized optimization methods. The main difficulty of the approach is to obtain an analytical description of the designed algorithms.

Analytical proofs Finally, efforts could also be made to derive analytical proofs of worst-case guarantees using the PEP techniques described in Section 3.4 which rely on the use of dual variables of the PEP.

About the development of decentralized optimization

I believe that the development of decentralized optimization should be focused on the elaboration of explainable, robust, and sufficiently efficient methods. The PEP techniques that we have developed in this thesis can facilitate progress in these directions.

Decentralization to democratize technologies The decentralization paradigm in algorithms gives decision-making power to local entities enabling them to cooperate in solving complex problems without the need for a large central entity. To ensure efficient cooperation among diverse agents, which may possess varying computing or communication capabilities and be owned by different individuals, it is important to have algorithms that are efficient in terms of computing power and memory usage. Additionally, these algorithms should be explainable to different individuals, so that they are willing to cooperate. Our new PEP tools can help with this, as they provide a better understanding of decentralized algorithms and their performance.

Rapidity versus robustness The rapidity of execution is far from being the only important measure of performance for a decentralized algorithm. Of course, fast algorithms generally expend fewer resources and enable a wider range of applications. But I think it is worth losing a little efficiency to gain in robustness. Indeed, in real world applications, it is often important to have algorithms that work in a variety of settings (different types of functions, communications, network topologies, etc.) and can adapt to many situations, including unexpected ones: computation errors, communication errors, agent failures, malicious agents, etc. These advanced situations are challenging to analyze theoretically but many of them could be analyzed using the PEP techniques we have developed in this thesis.

Beware of the rebound effect When an algorithm is made more efficient or accessible, its use may spread or increase, leading to greater overall resource expenditure. This phenomenon, known as the rebound effect, should be taken into account in the development of decentralized optimization, as we need to limit our resource usage to what is necessary to reduce our environmental impact. The risks of rebound effect in decentralized optimization include, for example:

- **Intensification**: when solving a problem becomes cheap, it may be solved more frequently than necessary.
- **Distributed cost**: while the resources used by one agent remain limited, the system may scale to a very large number of agents, increasing overall resource consumption.
- **Reallocation**: time and energy saved by algorithm execution might be redirected to other computations.
- **Creating needs**: the accessibility of decentralized optimization might create the need to embed electronic devices in many objects or systems.

Keeping such risks in mind when developing decentralized optimization is important to mitigate them, especially when moving from theory to applications. While decentralized optimization has great potential, we should pay attention to using it in applications where it matters and avoid deploying it at will.

List of Publications

⋆ Journal papers

[CH23] Sébastien Colla and Julien M. Hendrickx, "**Automatic Performance Estimation for Decentralized Optimization**", in IEEE Transactions on Automatic Control.

[CH24a] Sébastien Colla and Julien M. Hendrickx, "**Exploiting Agent Symmetries for Performance Analysis of Distributed Optimization Methods**", in Open Journal of Mathematical Optimization *(submitted)*.

⋆ Conference papers

[CH21] Sébastien Colla and Julien M. Hendrickx, "**Automated Worst-Case Performance Analysis of Decentralized Gradient Descent**", in Proceedings of the 60th IEEE Conference on Decision and Control (CDC 2022), Austin (Texas, USA), December 2021.

[CH22] Sébastien Colla and Julien M. Hendrickx, "**Automated Performance Estimation for Decentralized Optimization via Network Size Independent Problems**", in Proceedings of the 61th IEEE Conference on Decision and Control (CDC 2022), Cancun (Mexico), December 2022.

[RBCH23] Anne Rubbens, Nizar Bousselmi, Sébastien Colla and Julien M. Hendrickx, "**Interpolation Constraints for Computing Worst-Case Bounds in Performance Estimation Problems**", in Proceedings of the 62th IEEE Conference on Decision and Control (CDC 2023), Singapore, December 2023.

[CH24b] Sébastien Colla and Julien M. Hendrickx, "**On the Optimal Communication Weights in Distributed Optimization Algorithms**", in Proceedings of the 26th International Symposium on Mathematical Theory of Networks and Systems (MTNS 2024), Cambridge, August 2024. *(accepted)*

⋆ Posters

- "**Automatic Performance Estimation for Decentralized Optimization**", 7th International Conference on Continuous Optimization (ICCOPT), held in July 2022 in Lehigh University (USA).

⋆ Abstracts

- "**Decentralized Estimation in Open Multi-Agent Systems**", at the 19th European Control Conference (ECC), held online from June 29 to July 2, 2021.
- "**Automated Worst-Case Performance Analysis of Decentralized Optimization Methods**", at the Benelux Workshop on Systems and Control (2021), held on June 29, 2021 in Rotterdam (The Netherlands).
- "**Performance Estimation Problem for Decentralized Optimization Methods**", 41st Benelux Meeting on Systems and Control, held from July 5 to July 7, 2022 in Brussels (Belgium).
- "**A Size Independent Formulation for Automatic Performance Evaluation of Decentralized Optimization Algorithms**", 42th Benelux Meeting on Systems and Control, held from March 21 to July 23, 2023 in Elspeet (The Neederlands).
- "**What are the Optimal Communication Weights for Decentralized Optimization ?**", 43th Benelux Meeting on Systems and Control, held from March 26 to July 28, 2024 in Blankenberge (Belgium).

Bibliography

[ABC+20] Yossi Arjevani, Joan Bruna, Bugra Can, Mert Gurbuzbalaban, Stefanie Jegelka, and Hongzhou Lin. Ideal: Inexact decentralized accelerated augmented lagrangian method. *Advances in Neural Information Processing Systems*, 33:20648–20659, 2020.

[ACR08] Tuncer Can Aysal, Mark J Coates, and Michael G Rabbat. Distributed average consensus with dithered quantization. *IEEE transactions on Signal Processing*, 56(10):4905–4918, 2008.

[ADJ02] Charles Audet and John E Dennis Jr. Analysis of generalized pattern searches. *SIAM J. Optim.*, 13(3):889–903, 2002.

[AdKZ22] Hadi Abbaszadehpeivasti, Etienne de Klerk, and Moslem Zamani. Convergence rate analysis of randomized and cyclic coordinate descent for convex optimization through semidefinite programming. *arXiv preprint arXiv:2212.12384*, 2022.

[AdKZ23a] Hadi Abbaszadehpeivasti, Etienne de Klerk, and Moslem Zamani. Conditions for linear convergence of the gradient method for non-convex optimization. *Optimization Letters*, 17(5):1105–1125, 2023.

[AdKZ23b] Hadi Abbaszadehpeivasti, Etienne de Klerk, and Moslem Zamani. On the rate of convergence of the difference-of-convex algorithm (dca). *Journal of Optimization Theory and Applications*, pages 1–22, 2023.

[Ali95] Farid Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM journal on Optimization*, 5(1):13–51, 1995.

[AWLM17] Necdet Serhat Aybat, Zi Wang, Tianyi Lin, and Shiqian Ma. Distributed linearized alternating direction method of multipliers for composite convex consensus optimization. *IEEE Transactions on Automatic Control*, 63(1):5–20, 2017.

[AYS19] Sulaiman Alghunaim, Kun Yuan, and Ali H Sayed. A linearly convergent proximal gradient algorithm for decentralized optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[AYSS09] Tuncer Can Aysal, Mehmet Ercan Yildiz, Anand D Sarwate, and Anna Scaglione. Broadcast gossip algorithms for consensus. *IEEE Transactions on Signal processing*, 57(7):2748–2761, 2009.

[BBP19] Pintu Bhunia, Santanu Bag, and Kallol Paul. Bounds for eigenvalues of the adjacency matrix of a graph. *Journal of Interdisciplinary Mathematics*, 22, 06 2019. doi:10.1080/09720502.2019.1630938.

[BDX04] Stephen Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing markov chain on a graph. *SIAM Rev.*, 46(4):667–689, 2004.

[Ber83] Dimitri P Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983.

[BGPS06] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006.

[BHG23] Nizar Bousselmi, Julien M Hendrickx, and François Glineur. Interpolation conditions for linear operators and applications to performance estimation problems. *arXiv preprint arXiv:2302.08781*, 2023.

[BHOT05] Vincent D Blondel, Julien M Hendrickx, Alex Olshevsky, and John N Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proceedings of the 44th IEEE*

Conference on Decision and Control, pages 2996–3000. IEEE, 2005.

[BISV08] Guillermo Barrenetxea, François Ingelrest, Gunnar Schaefer, and Martin Vetterli. Wireless sensor networks for environmental monitoring: The sensorscope experience. In *2008 IEEE International Zurich Seminar on Communications*, pages 98–101. IEEE, 2008.

[BPC+11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3:1–122, 01 2011. doi:10.1561/2200000016.

[BPS+23] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Communications Surveys & Tutorials*, 2023.

[BS13] J Frédéric Bonnans and Alexander Shapiro. *Perturbation analysis of optimization problems*. Springer, 2013.

[BTB23] Mathieu Barré, Adrien B Taylor, and Francis Bach. Principled analyses and design of first-order methods with inexact proximal operators. *Mathematical Programming*, 201(1):185–230, 2023.

[BTd20] Mathieu Barré, Adrien Taylor, and Alexandre d'Aspremont. Complexity guarantees for polyak steps with momentum. In *Conference on learning theory*, pages 452–478. PMLR, 2020.

[CH21] Sebastien Colla and Julien M. Hendrickx. Automated worst-case performance analysis of decentralized gradient descent. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2627–2633, 2021. doi:10.1109/CDC45484.2021.9683505.

[CH22] Sebastien Colla and Julien M. Hendrickx. Automated performance estimation for decentralized optimization via net-

work size independent problems. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 5192–5199, 2022. doi:10.1109/CDC51059.2022.9993346.

[CH23] Sebastien Colla and Julien M. Hendrickx. Automatic performance estimation for decentralized optimization. *IEEE Transactions on Automatic Control*, 68(12):7136–7150, 2023. doi:10.1109/TAC.2023.3251902.

[CH24a] Sebastien Colla and Julien M Hendrickx. Exploiting agent symmetries for performance analysis of distributed optimization methods. *arXiv preprint arXiv:2403.11724*, 2024.

[CH24b] Sebastien Colla and Julien M Hendrickx. On the optimal communication weights in distributed optimization algorithms. In *26th International Symposium on Mathematical Theory of Networks and Systems (MTNS 2024)*, 2024.

[CHW+20] Tsung-Hui Chang, Mingyi Hong, Hoi-To Wai, Xinwei Zhang, and Songtao Lu. Distributed learning in the nonconvex world: From batch data to streaming and beyond. *IEEE Signal Processing Magazine*, 37(3):26–38, 2020.

[CI14] Kai Cai and Hideaki Ishii. Average consensus on arbitrary strongly connected digraphs with time-varying topologies. *IEEE Transactions on Automatic Control*, 59(4):1066–1071, 2014.

[CN22] Guido Carnevale and Giuseppe Notarstefano. Nonconvex distributed optimization via lasalle and singular perturbations. *IEEE Control Systems Letters*, 7:301–306, 2022.

[CO12] Annie I Chen and Asuman Ozdaglar. A fast distributed proximal-gradient method. In *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 601–608. IEEE, 2012.

[CYRC12] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438, 2012.

[CYY+15] Themistoklis Charalambous, Ye Yuan, Tao Yang, Wei Pan, Christoforos N Hadjicostis, and Mikael Johansson. Distributed finite-time average consensus in digraphs in the presence of time delays. *IEEE Transactions on Control of Network Systems*, 2(4):370–381, 2015.

[DA07] Nair Maria Maia De Abreu. Old and new results on algebraic connectivity of graphs. *Linear algebra and its applications*, 423(1):53–73, 2007.

[DAW12] John C. Duchi, Alekh Agarwal, and Martin J. Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic Control*, 57(3):592–606, 2012. doi:10.1109/TAC.2011.2161027.

[DGFST23] Shuvomoy Das Gupta, Robert M Freund, Xu Andy Sun, and Adrien Taylor. Nonlinear conjugate gradient methods: worst-case convergence rates via computer-assisted analyses. *arXiv preprint arXiv:2301.01530*, 2023.

[DGVPR24] Shuvomoy Das Gupta, Bart PG Van Parys, and Ernest K Ryu. Branch-and-bound performance estimation programming: A unified methodology for constructing optimal optimization methods. *Mathematical Programming*, 204(1):567–639, 2024.

[DKGT17] Etienne De Klerk, François Glineur, and Adrien B Taylor. On the worst-case complexity of the gradient method with exact line search for smooth strongly convex functions. *Optimization Letters*, 11:1185–1199, 2017.

[DKGT20] Etienne De Klerk, Francois Glineur, and Adrien B Taylor. Worst-case convergence analysis of inexact gradient and newton methods through semidefinite programming performance estimation. *SIAM Journal on Optimization*, 30(3):2053–2082, 2020.

[DKM+10] Alexandros G Dimakis, Soummya Kar, José MF Moura, Michael G Rabbat, and Anna Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864, 2010.

[DLS16] Paolo Di Lorenzo and Gesualdo Scutari. Next: In-network nonconvex optimization. *IEEE Transactions on Signal and Information Processing over Networks*, 2(2):120–136, 2016.

[DSK20] Amir Daneshmand, Gesualdo Scutari, and Vyacheslav Kungurtsev. Second-order guarantees of distributed gradient algorithms. *SIAM Journal on Optimization*, 30(4):3029–3068, 2020.

[DT12] Yoel Drori and Marc Teboulle. Performance of first-order methods for smooth convex minimization: A novel approach. *Mathematical Programming*, 145, 06 2012. doi:10.1007/s10107-013-0653-0.

[DT16] Yoel Drori and Marc Teboulle. An optimal variant of kelley's cutting-plane method. *Mathematical Programming*, 160:321–351, 2016.

[DT20] Yoel Drori and Adrien B Taylor. Efficient first-order methods for convex minimization: a constructive approach. *Mathematical Programming*, 184(1):183–220, 2020.

[DTdB22] Radu-Alexandru Dragomir, Adrien B Taylor, Alexandre d'Aspremont, and Jérôme Bolte. Optimal complexity and certification of bregman first-order methods. *Mathematical Programming*, pages 1–43, 2022.

[FCFZ09] Paolo Frasca, Ruggero Carli, Fabio Fagnani, and Sandro Zampieri. Average consensus on networks with quantized communication. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 19(16):1787–1816, 2009.

[FCG10] Pedro Forero, Alfonso Cano, and G.B. Giannakis. Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, 11:1663–1707, 05 2010.

[FH13] Paolo Frasca and Julien M Hendrickx. Large network consensus is robust to packet losses and interferences. In *2013 European Control Conference (ECC)*, pages 1782–1787. IEEE, 2013.

[Fie89] Miroslav Fiedler. Laplacian of graphs and algebraic connectivity. *Banach Center Publications*, 1(25):57–70, 1989.

[GBS08] Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM Rev.*, 50(1):37–66, 2008.

[GDT23] Baptiste Goujaud, Aymeric Dieuleveut, and Adrien Taylor. On fundamental proof structures in first-order optimization. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 3023–3030. IEEE, 2023.

[GEIGM22] Charles Guille-Escuret, Adam Ibrahim, Baptiste Goujaud, and Ioannis Mitliagkas. Gradient descent is optimal under lower restricted secant inequality and upper error bound. *Advances in Neural Information Processing Systems*, 35:24893–24904, 2022.

[GH18] Balázs Gerencsér and Julien M Hendrickx. Push-sum with transmission failures. *IEEE Transactions on Automatic Control*, 64(3):1019–1033, 2018.

[GKC17] Nirupam Gupta, Jonathan Katz, and Nikhil Chopra. Privacy in distributed average consensus. *IFAC-PapersOnLine*, 50(1):9515–9520, 2017.

[GLG22] Eduard Gorbunov, Nicolas Loizou, and Gauthier Gidel. Extragradient method: O (1/k) last-iterate convergence for monotone variational inequalities and connections with cocoercivity. In *International Conference on Artificial Intelligence and Statistics*, pages 366–402. PMLR, 2022.

[GMG+22] Baptiste Goujaud, Céline Moucer, François Glineur, Julien Hendrickx, Adrien Taylor, and Aymeric Dieuleveut. PEPit: computer-assisted worst-case analyses of first-order optimization methods in Python. *arXiv preprint arXiv:2201.04040*, 2022.

[GR01] Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2001.

[GRB+22] Eduard Gorbunov, Alexander Rogozin, Aleksandr Beznosikov, Darina Dvinskikh, and Alexander Gasnikov. Recent theoretical advances in decentralized distributed convex optimization. In *High-Dimensional Optimization and Probability: With a View Towards Data Science*, pages 253–325. Springer, 2022.

[GSS+19] Nicolo Gionfra, Guillaume Sandou, Houria Siguerdidjane, Damien Faille, and Philippe Loevenbruck. Wind farm distributed pso-based control for constrained power generation maximization. *Renewable energy*, 133:103–117, 2019.

[GTD22] Baptiste Goujaud, Adrien Taylor, and Aymeric Dieuleveut. Optimal first-order methods for convex functions with a quadratic upper bound. *arXiv preprint arXiv:2205.15033*, 2022.

[GTG22] Eduard Gorbunov, Adrien Taylor, and Gauthier Gidel. Last-iterate convergence of optimistic gradient method for monotone variational inequalities. *Advances in neural information processing systems*, 35:21858–21870, 2022.

[GTHG23] Eduard Gorbunov, Adrien Taylor, Samuel Horváth, and Gauthier Gidel. Convergence of proximal point and extragradient-based methods beyond monotonicity: the case of negative comonotonicity. In *International Conference on Machine Learning*, pages 11614–11641. PMLR, 2023.

[GY20] Guoyong Gu and Junfeng Yang. Tight sublinear convergence rate of the proximal point algorithm for maximal monotone inclusion problems. *SIAM Journal on Optimization*, 30(3):1905–1921, 2020.

[GY22] Guoyong Gu and Junfeng Yang. Tight ergodic sublinear convergence rate of the relaxed proximal point algorithm for monotone variational inequalities. *Journal of Optimization Theory and Applications*, pages 1–15, 2022.

[GYA18] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications (3rd ed.)*. Chapman and Hall/CRC, 2018.

[HBJ18] Lie He, An Bian, and Martin Jaggi. Cola: Decentralized linear learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[HHZ17] Mingyi Hong, Davood Hajinezhad, and Ming-Min Zhao. Prox-pda: The proximal primal-dual algorithm for fast distributed nonconvex optimization and learning over net-

works. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2017.

[HL05] Hanley and Chi-Kwong Li. Generalized doubly stochastic matrices and linear preservers. *Linear and Multilinear Algebra*, 53:1–11, 01 2005. doi:10.1080/03081080410001681599.

[HSYS21] Trevor Halsted, Ola Shorinwa, Javier Yu, and Mac Schwager. A survey of distributed optimization methods for multi-robot systems. *arXiv preprint arXiv:2103.12840*, 2021.

[IBCH15] Franck Iutzeler, Pascal Bianchi, Philippe Ciblat, and Walid Hachem. Explicit convergence rate of a distributed alternating direction method of multipliers. *IEEE Transactions on Automatic Control*, 61(4):892–904, 2015.

[IBS+10] François Ingelrest, Guillermo Barrenetxea, Gunnar Schaefer, Martin Vetterli, Olivier Couach, and Marc Parlange. Sensorscope: Application-specific sensor network for environmental monitoring. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):1–32, 2010.

[Jag11] Martin Jaggi. Convex optimization without projection steps. *arXiv preprint arXiv:1108.1170*, 2011.

[Jak18] Dušan Jakovetić. A unification and generalization of exact distributed first-order methods. *IEEE Transactions on Signal and Information Processing over Networks*, 5(1):31–46, 2018.

[JGR23] Uijeong Jang, Shuvomoy Das Gupta, and Ernest K Ryu. Computer-assisted design of accelerated composite optimization methods: Optista. *arXiv preprint arXiv:2305.15704*, 2023.

[JLM03] Ali Jadbabaie, Jie Lin, and A Stephen Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on automatic control*, 48(6):988–1001, 2003.

[JXM14] Dušan Jakovetić, Joao Xavier, and José MF Moura. Fast distributed gradient methods. *IEEE Transactions on Automatic Control*, 59(5):1131–1146, 2014.

[KBS15] Torben Knudsen, Thomas Bak, and Mikael Svenstrup. Survey of wind farm control—power and fatigue optimization. *Wind Energy*, 18(8):1333–1351, 2015.

[KDG03] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 482–491. IEEE, 2003.

[KF16] Donghwan Kim and Jeffrey A Fessler. Optimized first-order methods for smooth convex minimization. *Mathematical programming*, 159:81–107, 2016.

[KF17] Donghwan Kim and Jeffrey A Fessler. On the convergence analysis of the optimized gradient method. *Journal of optimization theory and applications*, 172(1):187–205, 2017.

[KF18] Donghwan Kim and Jeffrey A Fessler. Generalizing the optimized gradient method for smooth convex minimization. *SIAM Journal on Optimization*, 28(2):1920–1950, 2018.

[KF21] Donghwan Kim and Jeffrey A Fessler. Optimizing the efficiency of first-order methods for decreasing the gradient of smooth convex functions. *Journal of optimization theory and applications*, 188(1):192–219, 2021.

[KGGR21] Dmitry Kovalev, Elnur Gasanov, Alexander Gasnikov, and Peter Richtarik. Lower bounds and optimal algorithms for smooth and strongly convex decentralized optimization over time-varying networks. *Advances in Neural Information Processing Systems*, 34:22325–22335, 2021.

[KHG23] Yassine Kamri, Julien M Hendrickx, and François Glineur. On the worst-case analysis of cyclic coordinate-wise algorithms on smooth convex functions. In *2023 European Control Conference (ECC)*, pages 1–8. IEEE, 2023.

[KLB+20] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

[KM08] Soummya Kar and José MF Moura. Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise. *IEEE Transactions on Signal Processing*, 57(1):355–369, 2008.

[KSJ19] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International Conference on Machine Learning*, pages 3478–3487. PMLR, 2019.

[KSR20] Dmitry Kovalev, Adil Salim, and Peter Richtárik. Optimal and practical algorithms for smooth and strongly convex decentralized optimization. *Advances in Neural Information Processing Systems*, 33:18342–18352, 2020.

[KSR+21] Dmitry Kovalev, Egor Shulgin, Peter Richtárik, Alexander V Rogozin, and Alexander Gasnikov. Adom: Accelerated decentralized optimization method for time-varying networks. In *International Conference on Machine Learning*, pages 5784–5793. PMLR, 2021.

[LCZ22] Wei Liu, Li Chen, and Wenyi Zhang. Decentralized federated learning: Balancing communication and computing costs. *IEEE Transactions on Signal and Information Processing over Networks*, 8:131–143, 2022.

[LFYL20] Huan Li, Cong Fang, Wotao Yin, and Zhouchen Lin. Decentralized accelerated gradient methods with increasing penalty parameters. *IEEE transactions on Signal Processing*, 68:4855–4870, 2020.

[Lie21] Felix Lieder. On the convergence rate of the halpern-iteration. *Optimization letters*, 15(2):405–418, 2021.

[LJ11] Peng Lin and Yingmin Jia. Multi-agent consensus with diverse time-delays and jointly-connected topologies. *Automatica*, 47(4):848–856, 2011.

[LL20] Huan Li and Zhouchen Lin. Revisiting extra for smooth distributed optimization. *SIAM Journal on Optimization*, 30(3):1795–1821, 2020.

[LL21] Huan Li and Zhouchen Lin. Accelerated gradient tracking over time-varying graphs for decentralized optimization. *arXiv preprint arXiv:2104.02596*, 2021.

[LLKS24] Yue Liu, Tao Lin, Anastasia Koloskova, and Sebastian U Stich. Decentralized gradient tracking with local steps. *Optimization Methods and Software*, pages 1–28, 2024.

[LLST16] Qing Ling, Yaohua Liu, Wei Shi, and Zhi Tian. Weighted admm for fast decentralized network optimization. *IEEE Transactions on Signal Processing*, 64(22):5930–5942, 2016.

[LLZ20] Guanghui Lan, Soomin Lee, and Yi Zhou. Communication-efficient algorithms for decentralized and stochastic optimization. *Mathematical Programming*, 180(1):237–284, 2020.

[LMH18] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. Catalyst acceleration for first-order convex optimization: from theory to practice. *Journal of Machine Learning Research*, 18(212):1–54, 2018.

[LN15] Soomin Lee and Angelia Nedić. Asynchronous gossip-based random projection algorithms over networks. *IEEE Transactions on Automatic Control*, 61(4):953–968, 2015.

[LQZS12] Peng Lin, Kaiyu Qin, Hongmei Zhao, and Man Sun. A new approach to average consensus problems with multiple time-delays and jointly-connected topologies. *Journal of the Franklin Institute*, 349(1):293–304, 2012.

[LRP14] Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26, 08 2014. doi:10.1137/15M1009597.

[LRS16] Peng Lin, Wei Ren, and Yongduan Song. Distributed multi-agent optimization subject to nonidentical constraints and communication delays. *Automatica*, 65:120–131, 2016.

[LSY17] Zhi Li, Wei Shi, and Ming Yan. A decentralized proximal-gradient method with network independent step-sizes and separated convergence rates. *IEEE Transactions on Signal Processing*, PP, 04 2017. doi:10.1109/TSP.2019.2926022.

[LT10] Qing Ling and Zhi Tian. Decentralized sparse signal recovery for compressive sleeping wireless sensor networks. *Signal Processing, IEEE Transactions on*, 58:3816 – 3827, 08 2010. doi:10.1109/TSP.2010.2047721.

[LYH17] Qingshan Liu, Shaofu Yang, and Yiguang Hong. Constrained consensus algorithms with fixed step size for distributed convex optimization over multiagent networks. *IEEE Transactions on Automatic Control*, 62(8):4259–4265, 2017.

[LZ10] Tao Li and Ji-Feng Zhang. Consensus conditions of multi-agent systems with time-varying topologies and stochastic communication noises. *IEEE Transactions on Automatic Control*, 55(9):2043–2057, 2010.

[MBG10] Gonzalo Mateos, Juan Bazerque, and G.B. Giannakis. Distributed sparse linear regression. *Signal Processing, IEEE Transactions on*, 58:5262 – 5276, 11 2010. doi:10.1109/TSP.2010.2055862.

[MdGH22] Charles Monnoyer de Galland and Julien M Hendrickx. Fundamental performance limitations for average consensus in open multi-agent systems. *IEEE Transactions on Automatic Control*, 68(2):646–659, 2022.

[MDS+17] Daniel K Molzahn, Florian Dörfler, Henrik Sandberg, Steven H Low, Sambuddha Chakrabarti, Ross Baldick, and Javad Lavaei. A survey of distributed optimization and control algorithms for electric power systems. *IEEE Transactions on Smart Grid*, 8(6):2941–2962, 2017.

[MM16] Yilin Mo and Richard M Murray. Privacy preserving average consensus. *IEEE Transactions on Automatic Control*, 62(2):753–765, 2016.

[MMR+17] H. McMahan, Eider Moore, D. Ramage, S. Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[MO17] Ali Makhdoumi and Asuman Ozdaglar. Convergence rate of distributed admm over networks. *IEEE Transactions on Automatic Control*, 62(10):5082–5095, 2017.

[MR97] Alexandre Megretski and Anders Rantzer. System analysis via integral quadratic constraints. *IEEE transactions on automatic control*, 42(6):819–830, 1997.

[MR16] Aryan Mokhtari and Alejandro Ribeiro. Dsa: Decentralized double stochastic averaging gradient algorithm. *Journal of Machine Learning Research*, 17(61):1–35, 2016.

[MTB23] Céline Moucer, Adrien Taylor, and Francis Bach. A systematic approach to lyapunov analyses of continuous-time models in convex optimization. *SIAM Journal on Optimization*, 33(3):1558–1586, 2023.

[Ned10] Angelia Nedic. Asynchronous broadcast-based convex optimization over a network. *IEEE Transactions on Automatic Control*, 56(6):1337–1351, 2010.

[Nes18] Yurii Nesterov. *Lectures on convex optimization*, volume 137. Springer, 2nd edition, 2018.

[NL18] Angelia Nedić and Ji Liu. Distributed optimization for control. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:77–103, 2018.

[NO09] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54:48 – 61, 02 2009. doi:10.1109/TAC.2008.2009515.

[NO10] Angelia Nedić and Asuman Ozdaglar. Convergence rate for consensus with delays. *Journal of Global Optimization*, 47:437–456, 2010.

[NO14] Angelia Nedić and Alex Olshevsky. Distributed optimization over time-varying directed graphs. *IEEE Transactions on Automatic Control*, 60(3):601–615, 2014.

[NOOT09] Angelia Nedic, Alex Olshevsky, Asuman Ozdaglar, and John N Tsitsiklis. On distributed averaging algorithms and quantization effects. *IEEE Transactions on automatic control*, 54(11):2506–2517, 2009.

[NOP10] Angelia Nedic, Asuman Ozdaglar, and Pablo A Parrilo. Constrained consensus and optimization in multi-agent networks. *IEEE Transactions on Automatic Control*, 55(4):922–938, 2010.

[NOR18] A. Nedić, A. Olshevsky, and M. G. Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, 106(5):953–976, 2018. doi:10.1109/JPROC.2018.2817461.

[NOS16] Angelia Nedic, Alex Olshevsky, and Wei Shi. Achieving geometric convergence for distributed optimization over time-varying graphs. *SIAM Journal on Optimization*, 27, 07 2016. doi:10.1137/16M1084316.

[NOSU17] Angelia Nedić, Alex Olshevsky, Wei Shi, and César A Uribe. Geometrically convergent distributed optimization with uncoordinated step-sizes. In *2017 American Control Conference (ACC)*, pages 3950–3955. IEEE, 2017.

[NZC15] Seyedbehzad Nabavi, Jianhua Zhang, and Aranya Chakrabortty. Distributed optimization algorithms for wide-area oscillation monitoring in power systems using interregional pmu-pdc architectures. *IEEE Transactions on Smart Grid*, 6(5):2529–2538, 2015.

[Ols15] Alex Olshevsky. Linear time average consensus on fixed graphs. *IFAC-PapersOnLine*, 48(22):94–99, 2015.

[Ols17] Alex Olshevsky. Linear time average consensus and distributed optimization on fixed graphs. *SIAM Journal on Control and Optimization*, 55(6):3990–4014, 2017.

[OSM04] Reza Olfati-Saber and Richard M Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control*, 49(9):1520–1533, 2004.

[OT09] Alex Olshevsky and John N Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM journal on control and optimization*, 48(1):33–55, 2009.

[PBB11] Fabio Pasqualetti, Antonio Bicchi, and Francesco Bullo. Consensus computation in unreliable networks: A system theoretic approach. *IEEE Transactions on Automatic Control*, 57(1):90–104, 2011.

[PKP06] Joel Predd, S.B. Kulkarni, and H. Vincent Poor. Distributed learning in wireless sensor networks. *Signal Processing Magazine, IEEE*, 23:56–69, 07 2006. doi:10.1109/MSP.2006.1657817.

[PN21] Shi Pu and Angelia Nedić. Distributed stochastic gradient tracking methods. *Mathematical Programming*, 187(1):409–457, 2021.

[PR24] Chanwoo Park and Ernest K Ryu. Optimal first-order algorithms as a function of inequalities. *Journal of Machine Learning Research*, 25, 2024.

[PSXN20] Shi Pu, Wei Shi, Jinming Xu, and Angelia Nedić. Push–pull gradient methods for distributed optimization in networks. *IEEE Transactions on Automatic Control*, 66(1):1–16, 2020.

[QL17] Guannan Qu and Na Li. Harnessing smoothness to accelerate distributed optimization. *IEEE Transactions on Control of Network Systems*, 5(3):1245–1260, 2017.

[QL19] Guannan Qu and Na Li. Accelerated distributed nesterov gradient descent. *IEEE Transactions on Automatic Control*, 65(6):2566–2581, 2019. doi:10.1109/TAC.2019.2937496.

[RAW17] Minghao Ruan, Muaz Ahmad, and Yongqiang Wang. Secure and privacy-preserving average consensus. In *Proceedings of the 2017 workshop on cyber-physical systems security and privacy*, pages 123–129, 2017.

[RBCH23] Anne Rubbens, Nizar Bousselmi, Sebastien Colla, and Julien M Hendrickx. Interpolation constraints for computing worst-case bounds in performance estimation problems. *preprint*, 2023.

[RGP22] Teodor Rotaru, François Glineur, and Panagiotis Patrinos. Tight convergence rates of the gradient method on smooth

hypoconvex functions. *arXiv preprint arXiv:2203.00775*, 2022.

[RH24] Anne Rubbens and Julien M. Hendrickx. A constraint-based approach to function interpolation, with application to performance estimation for weakly convex optimisation. 2024.

[RLG+21] Alexander Rogozin, Vladislav Lukoshkin, Alexander Gasnikov, Dmitry Kovalev, and Egor Shulgin. Towards accelerated rates for distributed optimization over time-varying networks. In *Optimization and Applications: 12th International Conference, OPTIMA 2021, Petrovac, Montenegro, September 27–October 1, 2021, Proceedings 12*, pages 258–272. Springer, 2021.

[RN04] Michael Rabbat and Robert Nowak. Distributed optimization in sensor networks. pages 20–27, 04 2004. doi:10.1109/IPSN.2004.1307319.

[RN05] Michael G Rabbat and Robert D Nowak. Quantized incremental algorithms for distributed optimization. *IEEE Journal on Selected Areas in Communications*, 23(4):798–808, 2005.

[RTBG20] Ernest K Ryu, Adrien B Taylor, Carolina Bergeling, and Pontus Giselsson. Operator splitting performance estimation: Tight contraction factors and optimal parameter selection. *SIAM Journal on Optimization*, 30(3):2251–2271, 2020.

[RW10] Ram Rajagopal and Martin J Wainwright. Network-based consensus averaging with general noisy channels. *IEEE Transactions on Signal Processing*, 59(1):373–385, 2010.

[SBB+17] Kevin Scaman, Francis Bach, Sébastien Bubeck, Yin Tat Lee, and Laurent Massoulié. Optimal algorithms for smooth and strongly convex distributed optimization in networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3027–3036. PMLR, 2017.

[SBB+18] Kevin Scaman, Francis Bach, Sébastien Bubeck, Laurent Massoulié, and Yin Tat Lee. Optimal algorithms for non-smooth distributed optimization in networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[SBB+19] Kevin Scaman, Francis Bach, Sébastien Bubeck, Yin Tat Lee, and Laurent Massoulié. Optimal convergence rates for convex distributed optimization in networks. *Journal of Machine Learning Research*, 20(159):1–31, 2019.

[SC16] Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

[SL16] Ziqiang Shi and Rujie Liu. A better convergence analysis of the block coordinate descent method for large scale machine learning. *arXiv preprint arXiv:1608.04826*, 2016.

[SLWY14] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25, 04 2014. doi:10.1137/14096668X.

[SLWY15] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. A proximal gradient algorithm for decentralized composite optimization. *IEEE Transactions on Signal Processing*, 63(22):6013–6023, 2015.

[SLY+13] Wei Shi, Qing Ling, Kun Yuan, Gang Wu, and Wotao Yin. On the linear convergence of the admm in decentralized consensus optimization. *Signal Processing, IEEE Transactions on*, 62, 07 2013. doi:10.1109/TSP.2014.2304432.

[SN11] Kunal Srivastava and Angelia Nedic. Distributed asynchronous constrained stochastic optimization. *IEEE journal of selected topics in signal processing*, 5(4):772–790, 2011.

[SRNV10] S Sundhar Ram, Angelia Nedić, and Venugopal V Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications*, 147:516–545, 2010.

[SRNV12] S Sundhar Ram, A Nedić, and Venugopal V Veeravalli. A new class of distributed optimization algorithms: Application to regression of distributed data. *Optimization Methods and Software*, 27(1):71–88, 2012.

[SS19] Gesualdo Scutari and Ying Sun. Distributed nonconvex constrained optimization over time-varying digraphs. *Mathematical Programming*, 176:497–544, 2019.

[SSD22] Ying Sun, Gesualdo Scutari, and Amir Daneshmand. Distributed optimization based on gradient tracking revisited: Enhancing convergence rate via surrogation. *SIAM Journal on Optimization*, 32(2):354–385, 2022.

[SSPY23] Zhuoqing Song, Lei Shi, Shi Pu, and Ming Yan. Optimal gradient tracking for decentralized optimization. *Mathematical Programming*, pages 1–53, 2023.

[SSPY24] Zhuoqing Song, Lei Shi, Shi Pu, and Ming Yan. Provably accelerated decentralized gradient methods over unbalanced directed graphs. *SIAM Journal on Optimization*, 34(1):1131–1156, 2024.

[SVSL20] Akhil Sundararajan, Bryan Van Scoy, and Laurent Lessard. Analysis and design of first-order distributed optimization algorithms over time-varying graphs. *IEEE Transactions on Control of Network Systems*, 7(4):1597–1608, 2020.

[SZ03] Jos F. Sturm and Shuzhong Zhang. On cones of nonnegative quadratic functions. *Mathematics of Operations Research*, 28(2):246–267, 2003. URL: http://www.jstor.org/stable/4127019.

[Tay17] Adrien Taylor. *Convex interpolation and performance estimation of first-order methods for convex optimization*. PhD thesis, UCLouvain, 2017. Advisors : François Glineur and Julien M. Hendrickx.

[TB19] Adrien Taylor and Francis Bach. Stochastic first-order methods: non-asymptotic and computer-aided analyses via potential functions. In *Conference on Learning Theory*, pages 2934–2992. PMLR, 2019.

[TBA86] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[TD23] Adrien Taylor and Yoel Drori. An optimal gradient method for smooth strongly convex minimization. *Mathematical Programming*, 199(1):557–594, 2023.

[THG15a] A. B. Taylor, J. M. Hendrickx, and F. Glineur. Exact worst-case performance of first-order methods for composite convex optimization. *SIAM Journal on Optimization*, 27, 12 2015. doi:10.1137/16M108104X.

[THG15b] A. B. Taylor, J. M. Hendrickx, and F. Glineur. Smooth strongly convex interpolation and exact worst-case performance of first-order methods. *Mathematical Programming*, 161, 02 2015. doi:10.1007/s10107-016-1009-3.

[THG17] A. B. Taylor, J. M. Hendrickx, and F. Glineur. Performance estimation toolbox (PESTO): Automated worst-case analysis of first-order optimization methods. In *IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1278–1283, 2017. doi:10.1109/CDC.2017.8263832.

[THG18] A. B. Taylor, J. M. Hendrickx, and F. Glineur. Exact worst-case convergence rates of the proximal gradient method for composite convex minimization. *Journal of Optimization Theory and Applications*, 178, 08 2018. doi:10.1007/s10957-018-1298-1.

[Tsi84] John N Tsitsiklis. *Problems in decentralized decision making and computation*. PhD thesis, Massachusetts Institute of Technology, 1984.

[TTM11] Håkan Terelius, Ufuk Topcu, and Richard M Murray. Decentralized multi-agent optimization via dual decomposition. *IFAC proceedings volumes*, 44(1):11245–11251, 2011.

[TU22] Mohammad Taha Toghani and César A Uribe. Scalable average consensus with compressed communications. In *2022 American Control Conference (ACC)*, pages 3412–3417. IEEE, 2022.

[TVSL18] Adrien Taylor, Bryan Van Scoy, and Laurent Lessard. Lyapunov functions for first-order methods: Tight automated convergence guarantees. In *International Conference on Machine Learning*, pages 4897–4906. PMLR, 2018.

[UBTG24] Manu Upadhyaya, Sebastian Banert, Adrien B Taylor, and Pontus Giselsson. Automated tight lyapunov analysis for first-order methods. *Mathematical Programming*, pages 1–38, 2024.

[ULGN20] César A Uribe, Soomin Lee, Alexander Gasnikov, and Angelia Nedić. A dual approach for optimal algorithms in distributed optimization over networks. In *2020 Information Theory and Applications Workshop (ITA)*, pages 1–37. IEEE, 2020.

[Ver20] Jean-Philippe Vergne. Decentralized vs. distributed organization: Blockchain, machine learning and the future of the digital platform. *Organization Theory*, 1(4), 2020.

[VFP24] Renato Vizuete, Paolo Frasca, and Elena Panteley. Sis epidemics on open networks: A replacement-based approximation, 2024. arXiv:2403.16727.

[Via08] CP Viazminsky. Necessary and sufficient conditions for a function to be separable. *Applied mathematics and computation*, 204(2):658–670, 2008.

[VMdGF+24] Renato Vizuete, Charles Monnoyer de Galland, Paolo Frasca, Elena Panteley, and Julien M Hendrickx. Trends and questions in open multi-agent systems. In *Hybrid and Networked Dynamical Systems: Modeling, Analysis and Control*, pages 219–252. Springer, 2024.

[VSL19] Bryan Van Scoy and Laurent Lessard. A distributed optimization algorithm over time-varying graphs with efficient gradient evaluations. *IFAC-PapersOnLine*, 52(20):357–362, 2019.

[VSL20] Bryan Van Scoy and Laurent Lessard. Systematic analysis of distributed optimization algorithms over jointly-connected networks. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3096–3101. IEEE, 2020.

[VSL21] Bryan Van Scoy and Laurent Lessard. The speed-robustness trade-off for first-order methods with additive gradient noise. *arXiv preprint arXiv:2109.05059*, 2021.

[VSL23] Bryan Van Scoy and Laurent Lessard. A tutorial on a lyapunov-based approach to the analysis of iterative optimization algorithms. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 3003–3008. IEEE, 2023.

[XB04] Lin Xiao and Stephen Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1), 2004.

[XBK07] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. Distributed average consensus with least-mean-square deviation. *Journal of parallel and distributed computing*, 67(1):33–46, 2007.

[XCGN22] Zhiwei Xu, Bing Chu, Hua Geng, and Xiaohong Nian. Distributed power optimization of large wind farms using admm for real-time control. *IEEE Transactions on Power Systems*, 37(6):4832–4845, 2022.

[XK17] Chenguang Xi and Usman A Khan. Dextra: A fast algorithm for optimization over directed graphs. *IEEE Transactions on Automatic Control*, 62(10):4980–4993, 2017.

[XMX19] Liang Xu, Yilin Mo, and Lihua Xie. Distributed consensus over markovian packet loss channels. *IEEE Transactions on Automatic Control*, 65(1):279–286, 2019.

[XPNK20] Ran Xin, Shi Pu, Angelia Nedić, and Usman A Khan. A general framework for decentralized optimization with first-order methods. *Proceedings of the IEEE*, 108(11):1869–1889, 2020.

[XTSS20] Jinming Xu, Ye Tian, Ying Sun, and Gesualdo Scutari. Accelerated primal-dual algorithms for distributed smooth convex optimization over networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2381–2391. PMLR, 2020.

[XTSS21] Jinming Xu, Ye Tian, Ying Sun, and Gesualdo Scutari. Distributed algorithms for composite optimization: Unified framework and convergence analysis. *IEEE Transactions on Signal Processing*, 69:3555–3570, 2021.

[XXK18] Chenguang Xi, Ran Xin, and Usman A. Khan. Add-opt: Accelerated distributed directed optimization. *IEEE Transactions on Automatic Control*, 63(5):1329–1339, 2018. doi:10.1109/TAC.2017.2737582.

[XZSX15] Jinming Xu, Shanying Zhu, Yeng Chai Soh, and Lihua Xie. Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 2055–2060. IEEE, 2015.

[YHL15] Peng Yi, Yiguang Hong, and Feng Liu. Distributed gradient algorithm for constrained optimization with application to load sharing in power systems. *Systems & Control Letters*, 83:45–52, 2015.

[YHX15] Deming Yuan, Daniel WC Ho, and Shengyuan Xu. Regularized primal–dual subgradient method for distributed constrained optimization. *IEEE transactions on cybernetics*, 46(9):2109–2118, 2015.

[YLW+16] Tao Yang, Jie Lu, Di Wu, Junfeng Wu, Guodong Shi, Ziyang Meng, and Karl Henrik Johansson. A distributed algorithm for economic dispatch over time-varying directed networks with delays. *IEEE Transactions on Industrial Electronics*, 64(6):5095–5106, 2016.

[YLY13] Kun Yuan, Qing Ling, and Wotao Yin. On the convergence of decentralized gradient descent. *SIAM Journal on Optimization*, 26, 10 2013. doi:10.1137/130943170.

[YLZZ23] Haishan Ye, Luo Luo, Ziang Zhou, and Tong Zhang. Multi-consensus decentralized accelerated gradient descent. *Journal of Machine Learning Research*, 24(306):1–50, 2023.

[YTX13] Shiping Yang, Sicong Tan, and Jian-Xin Xu. Consensus based approach for economic dispatch problem in a smart grid. *IEEE Transactions on Power Systems*, 28(4):4416–4426, 2013.

[YYW+19] Tao Yang, Xinlei Yi, Junfeng Wu, Ye Yuan, Di Wu, Ziyang Meng, Yiguang Hong, Hong Wang, Zongli Lin, and Karl H

Johansson. A survey of distributed optimization. *Annual Reviews in Control*, 47:278–305, 2019.

[ZAdK23] Moslem Zamani, Hadi Abbaszadehpeivasti, and Etienne de Klerk. The exact worst-case convergence rate of the alternating direction method of multipliers. *Mathematical Programming*, pages 1–34, 2023.

[ZM10] Minghui Zhu and Sonia Martínez. Discrete-time dynamic average consensus. *Automatica*, 46(2):322–329, 2010.

[ZM11] Minghui Zhu and Sonia Martinez. On distributed convex optimization under inequality and equality constraints. *IEEE Transactions on Automatic Control*, 57(1):151–164, 2011.