Optimization methods over networks for popular content delivery and distributed machine learning

by

Marina Costantini

a dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

at the

Ecole Doctorale Informatique, Télécommunications et Electronique

of

Sorbonne Université

Defense scheduled on June 26th 2023 before a committee composed of:

Thesis supervisor	
Thrasyvoulos Spyropoulos	Technical University of Crete & EURECOM
Reviewers	
Marco Lorenzi	Inria & Université Côte d'Azur
Olivier Fercoq	Télécom Paris
President of the jury	
Pietro Michiardi	EURECOM
Examiners	
Aurélien Bellet	Inria & Université de Lille
Angelia Nédich	Arizona State University
Giovanni Neglia	Inria & Université Côte d'Azur

A scientist in his (her) laboratory is not a mere technician: he (she) is also a child confronting natural phenomena that impress him (her) as though they were fairy tales.
— Attributed to Marie Curie

“Happy who like Ulysses has done a beautiful journey”.
Stone mosaic at the top of the Colline du Château, Nice, France.

(Once again,) to my parents.
For their unconditional love, their constant support,
and for having given me, since always, all the means to be happy;
including the education that allowed me to be here today.

And to Apostolos.
For creating some of the best moments of my life and
standing by my side during some of the worst.
For bringing me happiness daily.

Words of thanks

I am very fortunate to have many family members, friends, and acquaintances that continuously contribute to my happiness and well-being, and therefore, who have indirectly contributed to the realization of this thesis. However, I will limit myself to thank here those who have had the most direct impact on its completion: those who made its quality significantly higher, and those without whom it would not have been realized at all.

First of all, my thanks go to Akis, my supervisor. For his great patience, his support, and his encouragement. For the time and thought that he has dedicated to our work. For making me leave every meeting with a more optimistic view than I had when I walked in, and for bringing me back to earth when enthusiasm was blinding me from all the work remaining to do. For pushing me to strive for excellence. And in addition, for sharing (and often feeding!) my nerd humor. I couldn't have asked for a better supervisor.

To Nikos Liakopoulos, my optimization mentor, collaborator, and dear, dear friend. For joining us in this adventure and for his guidance. For reviewing thoroughly (and sometimes decoding) my writing. For his listening and his advice. For discussing where we would go for drinks when working, and for discussing exciting work while having drinks. And for always reminding me that είμαι μία μπουλντόζα.

To Panayotis Mertikopoulos and Giovanni Neglia, other two excellent collaborators whose challenging questions and great suggestions have made the work presented here much stronger, and from whom I have learned a great amount.

To Georges Rodriguez and Roxana Ojeda, who early on pointed out to me the existence of a crucial node in the path that brought me here in the decision tree of life. We don't get to search this massive tree, and still, this path feels just optimal.

To my parents, Roberto and Cecilia, who have witnessed this long journey from start to end and have accompanied me at each step despite the distance. While shorter than the journey of Ulysses, mine was still full of challenges and hard decisions which I could face largely thanks to the understanding and support of my parents. To them I partially dedicate this thesis, for to them I owe the privilege of being able to have ambitious dreams and to go after them.

Last, but most definitely not least, to Apostolos, my partner in life. A large part of the work for this thesis took place during the dystopian times of the covid-19 pandemic. This brought us to work (literally) side by side, and all this time spent together, together with his affection and encouragement, has helped me to work more happily, overcome many limiting thoughts, and attain what I once thought was out of reach. This thesis is in part dedicated to him, without whom I couldn't have achieved all of this.

Contents

List of Figures

List of Tables

List of Acronyms

Here is a list of acronyms used in the text, in order of appearance.

Chapter 2

iid	Independent and identically distributed
SC	Small cell
CHR	Cache Hit Ratio
JCR	Joint Caching and Recommendation algorithm
SCH	Soft Cache Hits algorithm
POP	Popularity caching algorithm
ER	Erdős–Rényi graph
BA	Barabási–Albert graph
SVM	Support vector machine classifier

Chapter 3

CD	Coordinate descent
SU-CD	Set-wise uniform coordinate descent algorithm
SGS-CD	Set-wise Gauss-Southwell coordinate descent algorithm
SL-CD	Set-wise Lipschitz coordinate descent algorithm
SGSL-CD	Set-wise Gauss-Southwell Lipschitz coordinate descent algorithm
SeL-CD	SL-CD algorithm with estimated L_ℓ constants
SGSeL-CD	SGS-CD algorithm with estimated L_ℓ constants

Chapter 4

FL	Federated learning
PD	Parallel distributed computing
SGD	Stochastic rgadient descent

Chapter 1

Introduction

The ever-increasing number of users and applications on the Internet sets many challenges for network operators and engineers in order to keep up with the high traffic demands. Indeed, not only the number of people using the Internet has grown abruptly, going from 7% of the global population in 2000 to 60% today [1], but so did the number of connected devices per capita, going from 2.4 in 2018 to 3.6 currently [2]. In this scenario, making efficient use of the resources available has become imperative. In this thesis, we develop optimization methods to improve the utilization of the network in two specific applications enabled by the Internet: network edge caching and distributed model training.

Network edge caching is a recent technique that proposes to store at the network edge copies of contents that have a high probability of being requested. The motivation lies on the fact that bringing popular content geographically close to the end-users can reduce latency and improve the overall user experience. Traditionally, when a user requests a web page or application, the request is sent to a remote server that stores the data. The server retrieves the requested data and sends it back to the user. This process can be slow and can lead to latency and congestion issues, especially when multiple users are accessing the same data simultaneously.

To address this issue, network operators can deploy edge caching servers at strategic locations within their networks, close to the end-users. During off-peak hours, frequently accessed content (images, audio, videos) is moved to these servers so that when a user issues a request, the content is retrieved from the nearest edge caching server containing the content. This allows the request to be satisfied in a short time and high quality independently of the traffic conditions of the backhaul network.

On the other hand, distributed model training, or more generally, distributed optimization, is a method for training large-scale machine learning models using multiple computational *agents* that work together to find the optimal parameters of the model. In this thesis, we consider settings where the data at each agent are different, and agents interleave local computation steps and communication steps (either with other agents or with a central coordinator) to train a model that takes into account the data of all agents. To achieve this, agents may communicate optimization values (parameters, gradients) but *not* the data. Such a setting may be found in a computing cluster, where

data can be distributed among the servers either to speed up the training process by parallelization, or due to storage limitations at each individual server. Another typical use case is collaborative model training at the network edge, where multiple entities are interested in the same model but do not want to share their data due to privacy reasons. Examples are mobile users who want to have better-performing applications (e.g. next-word keyboard prediction [3]), health centers that are eager to collaborate to train more accurate models [4–6], or industrial applications that require potentially sensitive client data [7–9].

Here we consider two such distributed training settings: the decentralized and the federated. In the decentralized setting, agents are interconnected in a network and communicate their optimization values only to their direct neighbors. In the federated, the agents communicate with a central server that regularly averages the most recent values of (usually a subset of) the agents and broadcasts the result to *all* of them. Naturally, the success of such techniques relies on the frequent communication of the agents between them or with the server. Furthermore, in order to reach a certain accuracy, both the number of communication rounds and the amount of data transmitted at each round increase with the complexity of the model (i.e., with its number of parameters). There is thus great interest in designing distributed optimization algorithms that achieve state-of-the-art performance at lower communication costs.

In this thesis, we propose algorithms that improve the performance of existing methods for popular content delivery and distributed machine learning by making a better utilization of the network resources. In Chapter 2, we propose an algorithm that exploits recommendation engines to design jointly the contents cached at the network edge and the recommendations shown to the user. This algorithm achieves a higher fraction of requests served by the cache than its competitors and thus requires less communication with the remote server. In Chapter 3, we design an asynchronous algorithm for decentralized optimization that requires minimum coordination between the agents and thus partially mitigates the problem of leaving fast nodes idle while slow nodes complete their updates. We then show that, if the agents are allowed to increase the amount of data they transmit by a factor equal to their node degree, the convergence of this algorithm can potentially be made much faster by letting the agents decide their communication scheme according to the gains provided by communicating with each of their neighbors. Finally, in Chapter 4 we propose an algorithm that exploits inter-agent communication within the classical federated learning setup (where, in principle, agents communicate only with the server), and which can achieve the same convergence speed as the classical setup with much fewer communication rounds with the server, which constitute the main bottleneck in this setting.

In the next section, we give a more detailed overview of the contents of each of these chapters with an emphasis on our technical contributions in each case.

1.1 Outline of the thesis and contributions

1.1.1 Chapter 2: Joint design of caching and recommendations

Background

Caching popular content at the network edge can benefit both the network operator and the users by alleviating the backhaul traffic and reducing access latency, respectively. A successful caching system is one that satisfies as many requests as possible with the contents stored at the network edge, so that the amount of data retrieved from the remote server is minimized. The fraction of the total requests satisfied with the edge cache is called *cache hit ratio* (CHR), and is a classical measure of the quality of a caching system. Most of the caching literature [10–14] proposes algorithms that choose a few items to maximize the CHR under the constraint that their total size does not exceed the capacity of the cache, whose size is orders of magnitude smaller than that of the complete content catalog.

Recently, major content providers like Netflix and Google started partnering with Internet service providers to implement their own content distribution network solutions inside the network [15, 16]. This has opened a new possibility for the design of both the content to cache and the recommendations for the user, since now the same entity can control both. This seems to be a great opportunity to improve the performance of caching systems, since it is reported that up to 80% of requests on popular content distribution platforms (Netflix, Spotify, YouTube) come from user recommendations [17, 18]. Overall, this suggests that designing caching and recommendation policies separately is suboptimal: caching could benefit by knowing the recommender's actions in advance, and recommendation algorithms could try to favor cached content (among equally interesting options) to improve network performance and user experience.

In this chapter, we tackle the problem of optimally making caching and recommendation decisions jointly, in the context of the recently introduced "soft cache hits" [19] setup. A *soft hit* occurs when the content requested by the user is not found in the cache but the user accepts an alternative content that is offered by the system and *is* in the cache. This is in contrast with the standard *direct hits*, which occur when the content requested is in the cache. Previous work considering soft cache hits assumed that *any* number of contents stored in the cache may be offered as substitutes, and potentially the whole content of the cache. This is an unrealistic assumption, since although the cache capacity is much smaller than the total size of the catalog, it can still contain many more contents than what is practical to display to a user when browsing. Instead, here we consider that the number of substitute contents offered, i.e. the *recommendations*, is constrained and much smaller than the capacity of the cache. As we discuss next, this additional constraint complicates significantly the problem, and this increased complexity constitutes the base for our contributions.

Technical contributions

We start by showing that the problem of designing jointly the caching and recommendation decisions is NP-hard. This is a direct consequence of the fact that the original soft cache hits framework [19], where only the caching variables are considered, is already NP-hard by reduction to the maximum coverage problem with weighted elements.

However, we show that the joint caching and recommendation problem can be decomposed into an inner (recommendation-related) problem and an outer (caching-related) problem. The inner problem can be solved to optimality in polynomial time, i.e. for a given set of caching decisions, we can find the optimal recommendations for each potential request efficiently. Furthermore, we show that once decomposed, the complete problem is submodular in the caching decisions. This allows us to define an iterative greedy algorithm that runs in polynomial time and has approximation guarantees to the optimal solution.

We test the performance of the proposed algorithm in numerical simulations using both synthetic and real-world datasets, and demonstrate that it outperforms the competitors in all cases. Furthermore, for the real-world datasets considered, we observe a high variability in (i) the fraction of CHR accrued through direct hits versus soft hits, (ii) the performance ranking of the algorithms considered (although our proposal always ranks first), and (iii) the magnitude of CHR by which our algorithm outperforms the competitors.

These observations motivate us to investigate further *what* features of the datasets and the system setup affect the performance of caching and recommendation policies, and *how* they do so. For this, we model the similarity relations that make some contents good substitutes for some others as a graph (denoted *content graph*), where each content is a node and related contents are linked by a weighted edge. We then consider a set of features representative of the graph structure (degree skewness, clustering coefficient, Newman's modularity) and of the caching system design (cache size, number of recommendations) and study how these features impact on the absolute and relative performance of (i) our proposed caching and recommendation algorithm and (ii) a simple baseline that caches the most popular contents and chooses the recommendations a posteriori.

Our results confirm that features such as the degree distribution and the clustering coefficient of the graph are crucial to decide whether our proposed algorithm will perform significantly better than the simple baseline. Furthermore, the impact of some features is tightly related to the values of others, e.g. community structure and cache size, or popularity skewness and number of recommendations.

Motivated by these observations, we next address the question of whether these features related to the content graph and system setup could be used to predict the gains attainable by each of the tested algorithms. Our goal with this experiment is to test whether we could train a predictor to decide, based on these features, whether significant CHR gains could be obtained by our joint caching and recommendation algorithm compared to a simple baseline. Indeed, even polynomial algorithms can be prohibitively slow for large catalog sizes. Therefore, having such a classifier could help the system operators decide whether it is worth running a relatively complex algorithm for the joint caching and

recommendation design, or a simple baseline will do just as well. Our results show that with careful training of the classifier and a sufficiently representative dataset it is indeed possible to get good predictions.

The contributions of this chapter appear in:

• Costantini, M., Spyropoulos, T., Giannakas, T., & Sermpezis, P, "Approximation guarantees for the joint optimization of caching and recommendation," in *2020 IEEE International Conference on Communications (ICC) (pp. 1-7).*

• Costantini, M., & Spyropoulos, T., "Impact of popular content relational structure on joint caching and recommendation policies," in *2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT) (pp. 1-8).*

1.1.2 Chapter 3: Local Gauss-Southwell rule for decentralized optimization

Background

Many timely applications require solving optimization problems over a network where nodes can only communicate with their direct neighbors. This may be due to the need of distributing storage and computation loads (e.g. training large machine learning models [20]), or to avoid transferring data that is naturally collected in a decentralized manner, either due to the communication costs or to privacy reasons (e.g. sensor networks [21], edge computing [22], and precision medicine [23]).

We consider a setting where nodes interleave local computation steps and communication steps with their direct neighbors to collaboratively solve an optimization problem given by

$$\underset{\theta\in\mathbb{R}^d}{\text{minimize}} \quad \sum_{i=1}^{n} f_i(\theta), \tag{1.1}$$

where n is the number of nodes, each local function f_i is known only by node i, and nodes can exchange optimization values (parameters, gradients) but *not* the local functions themselves. This setting was first introduced in [24] and it has been extensively studied ever since. Contributions in this field include the design of algorithms with faster convergence guarantees [25–28], with reduced communication overhead [29, 30], or tailored to specific settings such as directed-communication [31, 32] or time-changing networks [33, 34], to name only a few.

In this setting, most works consider synchronous updates where all nodes must complete their local computations and have exchanged their values with their neighbors before a new iteration can take place [26–29, 33–38]. However, synchronizing the updates of all nodes can entail a far from optimal use of the resources, since (i) the control messages needed to coordinate the updates increase the congestion of the network, and (ii) if nodes

have different computing speeds (known as *systems heterogeneity*) fast nodes may stay idle for large periods of time waiting for the slow nodes (known as *stragglers*) before starting a new update.

In order to reduce such problems, much of the recent literature has focused on the analysis and design of asynchronous decentralized optimization algorithms where nodes can activate anytime and contact a single neighbor to complete an iteration together [25, 31, 39–42]. However, most of these works assume that when a node activates, it simply selects the neighbor to contact randomly, based on a predefined probability distribution. This approach overlooks the possibility of letting nodes *choose* the neighbor to contact taking into account the optimization landscape at the time of activation.

In contrast, in this chapter we propose to exploit the extra degree of freedom of asynchronous algorithms given by the neighbor choice to speed up convergence. In particular, we propose an algorithm where at each iteration the agents select the neighbor yielding the maximum suboptimality reduction. While this selection requires increasing the number of neighbors queried each time a node activates, we show remarkable gains in the progress made at each iteration. This indicates that, in settings where the nodes and the communication links are not working at capacity and their load can be increased, our algorithm can achieve the target accuracy in less wall-clock time than its random neighbor sampling counterpart.

Technical contributions

We consider an asynchronous decentralized setting where nodes can activate at any time and choose a neighbor to complete an iteration. To design algorithms that can run under such scheme, we depart from a reformulation of (1.1) as the constrained problem:

$$\begin{aligned} \underset{\theta_1,\dots,\theta_n \in R^d}{\text{minimize}} \quad & \sum_{i=1}^{n} f_i(\theta_i) \\ \text{s.t.} \quad & \theta_i = \theta_j \quad \forall\, (i,j) \equiv \ell \in \mathcal{E}, \end{aligned} \tag{1.2}$$

where $\mathcal{E}$ is the set of communication links (with size $|\mathcal{E}| = E$) and $\ell \equiv (i,j)$ indicates that $\ell \in \mathcal{E}$ connects nodes i and j. We propose to find the solution to (1.2) by solving its dual problem, which allows us to define an asynchronous algorithm closely related to coordinate descent methods, as explained next.

When going to the dual problem, instead of having one variable $\theta_i, i \in [n]$ per node (cf. eq. (1.2)) we have one variable $\lambda_\ell, \ell \in [E]$ per edge. Therefore, given our asynchronous communication scheme, we define an algorithm where every time a node i activates and contacts a neighbor j, both nodes perform a gradient update to the dual variable $\lambda_\ell, \ell \equiv (i,j)$ corresponding to the edge that connects them. We show that the two nodes involved have all the information necessary to perform this update. Therefore, many pairs of nodes can simultaneously update their shared dual variables independently of what is happening in the rest of the network.

We then remark that such algorithm can be seen as a coordinate descent (CD) [43,44] algo-

rithm: indeed, if we concatenate the dual variables such that $\lambda = [\lambda_1, \dots, \lambda_E] \in \mathbb{R}^{Ed \times 1}$ is our dual parameter vector, the updates described above correspond to a CD algorithm that updates a single (vector, also called *block*) coordinate $\lambda_\ell \in \mathbb{R}^d$ at a time. This insight allows us to leverage a few results of classical CD theory and define two versions of the decentalized asynchronous protocol defined above: one based on random (uniform) sampling [43] for the coordinate (or in our case, neighbor) selection, and another one which instead applies the Gauss-Southwell rule [45, 46], which selects the coordinate (neighbor) that provides the maximum suboptimality reduction at that time.

There is, however, a crucial difference between our algorithm and standard (single-machine) CD methods: while in the latter *any* of the coordinates may be updated, in our algorithm *only a small subset of coordinates* is accessible at each step, which consists of the dual variables associated with the edges connected to the node activated. We call this generic family of algorithms *set-wise coordinate descent* algorithms (Sx-CD), and denote the two versions proposed above for our decentralized asynchronous setting set-wise uniform CD (SU-CD) and set-wise Gauss-Southwell CD (SGS-CD). We show that, despite the similarities with classical CD, proving linear convergence of the Sx-CD algorithms for smooth and strongly convex f_i is not straightforward. Indeed, the analysis is greatly hindered by the fact that (i) even if the node functions f_i are smooth and strongly convex, the dual function in general is not, and that (ii) the sets of coordinates accessible by each node are overlapping, since each edge is connected to two nodes, and thus its dual variable appears in two such sets.

Despite these difficulties, by means of carefully defining the norms in which the functions' strong convexity is measured in each case, we prove linear convergence rates for both SU-CD and SGS-CD. In particular, we show that in terms of number of iterations SGS-CD can be up to $N_{\max}$ times faster than SU-CD, where $N_{\max}$ is the size of the largest set of coordinates (or in the decentralized case, the largest degree in the network). These algorithms are not only suitable for (dual) decentralized optimization, but also for (primal) parallel distributed optimization with parameter server, for which we show that the $N_{\max}$ speedup of SGS-CD with respect to SU-CD can be achieved.

Inspired once again by results in CD literature, we further propose two SxCD algorithms that exploit the knowledge of the per-coordinate Lipschitz constants of the dual function, each of them based on SU-CD and SGS-CD, and called SL-CD and SGSL-CD respectively. We prove linear rates also for these algorithms, and we show how the convergence in terms of the number of iterations of all four Sx-CD versions compare: (i) as SGS-CD, also SL-CD is always equal or faster than SU-CD, (ii) we cannot decide which between SL-CD and SGS-CD may be faster, but (iii) SGSL-CD is the fastest of them all. As with SU-CD and SGS-CD, proving the rates of the Lipschitz-informed versions required defining carefully the norms in which to measure strong convexity and also keeping the bounds tight for the comparison between algorithms.

Additionally, we propose an algorithm based on backtracking [43] to estimate the per-coordinate Lipschitz constants when they are not known a priori. We show that the versions of SL-CD and SGSL-CD using this algorithm, called SeL-CD and SGSeL-CD respectively, can achieve a remarkable performance in terms of number of iterations, at the cost of an increased amount of communication and computation.

We verify in numerical simulations that the relative performance of the algorithms (in terms of the number of iterations) agrees with our theoretical results. We also analyze their convergence in terms of the number of vectors in $\mathbb{R}^d$ transmitted through the network, which provides additional insight: under this view, the performance of the algorithms using the GS rule is much closer to those performing random sampling, and the latter actually take the lead place as the network becomes more connected. This is expectable, since the communication complexity of SGS-CD and SGSL-CD grows directly with the degree of the nodes, while that of SU-CD and SL-CD remains unchanged. However, we remark that neither the convergence of the algorithms in terms of the number of iterations nor of the number of vectors transmitted gives a complete picture to decide which algorithm will be faster in terms of wall-clock time: since the algorithms are asynchronous and can modify multiple coordinates simultaneously, in a real setup many iterations and vector transmissions will occur at the same time. We expect therefore that for networks that are not too densely connected, if the system is not working at capacity and the extra computation and communication costs of the Sx-CD algorithms using the GS rule can be accommodated without exceeding the capacity, these will converge faster in terms of wall-clock time than their randomized variants.

The contributions of this chapter appear in:

• Costantini, M., Liakopoulos, N., Mertikopoulos, P., & Spyropoulos, T., "Pick your Neighbor: Local Gauss-Southwell Rule for Fast Asynchronous Decentralized Optimization," in *2022 IEEE 61st Conference on Decision and Control (CDC) (pp. 1602-1609). IEEE.*

• Costantini, M., Liakopoulos, N., Mertikopoulos, P., & Spyropoulos, T., "Set-wise Coordinate Descent for Dual Asynchronous Decentralized Optimization," to be submitted to *IEEE Transactions on Automatic Control.*

1.1.3 Chapter 4: Peer-to-peer aided federated learning

Background

Federated learning (FL) is a recent machine learning framework that allows multiple agents, each of them with their own dataset, to train a model collaboratively without sharing their data. While this is similar to other distributed settings such as decentralized optimization or parallel distributed computing, the FL setting has three distinctive characteristics: (i) high data heterogeneity among the agents (both very non-iid and with large differences in dataset size), (ii) a massive number of participating agents, and (iii) limited communication due to unreliable connections.

Given these constraints, the authors of [47] proposed FederatedAveraging (FedAvg), the first and currently most popular FL algorithm, whose steps consist on the following:

1. The agents run H epochs of mini-batch stochastic gradient descent.
2. The server selects K agents at random and computes the weighted average of their current parameters, where the weight is proportional to the amount of data of each

selected agent.
3. The server broadcasts the new value to all agents, who set this as their current parameter value. The process then repeats from step 1.

Despite the excellent performance of FedAvg in simulations, the authors of [47] did not provide a theoretical analysis of the algorithm in the paper. This set off the distributed learning community to try to confer convergence guarantees to FedAvg, sometimes with slight modifications of the algorithm [48–53]. Indeed, FedAvg may not even converge when the number of stragglers and/or the data heterogeneity are high [48].

A fundamental trade-off in FL (as in most distributed optimization algorithms) is communication cost versus convergence speed: if the communication rounds with the server become more spaced in time (i.e., if H increases), convergence slows down [50, 54]. In this chapter, we propose an algorithm that exploits inter-agent communication to allow for more infrequent server communication rounds (i.e., larger H) without slowing down convergence, or equivalently, we show that for a fixed H, inter-agent communication leads to faster convergence. Moreover, we show that the magnitude of this speedup is directly related to the connectivity of the network. Our proposal can be thought of as *redistributing* the communication load in the FL setting by allocating a larger part of the spectrum to inter-agent communication and a smaller part to server communication rounds, which can now happen more infrequently without hurting convergence.

Technical contributions

We consider an FL setting where the agents can communicate with the server and between them. We assume that the agent-server communication is constrained by the very large number of devices and the limited bandwidth, which results in infrequent server communication rounds and partial device participation (i.e., only a few devices are sampled by the server in each server communication round). On the other hand, since (i) each agent is expected to have much fewer neighbors than the total number of agents, and (ii) short-range inter-agent communications allow for spectrum reuse, agents can communicate much more often between them than with the server [55, 56].

In this setting, we propose to interleave neighbor communication and parameter averaging in between the stochastic gradient descent (SGD) local updates of FedAvg. In other words, after every (mini-batch) SGD step taken at an agent, our algorithm lets the agent exchange its new parameter with its neighbors and perform a weighted average of all the received values and its own before the next SGD step. We additionally assume that due to fluctuations of the communication channel, the agents may communicate with only a subset of its neighbors at each time. We call this algorithm FedDec, since it can be thought of as a mixture between FL and (primal) decentralized optimization updates.

We analyze the convergence of FedDec under the assumptions of non-iid data distribution, partial device participation, and smooth and strongly convex costs. We show that for a diminishing stepsize, FedDec converges at the $O(1/T)$ rate (where T is the total number of iterations executed) of FL algorithms that do not account for inter-agent communication [50, 54], but improves the dependence on the number of local updates H from $O(H^2)$ to

$O(H)$. This improvement can be seen from two different points of view: either (i) H can be increased without severely hurting convergence speed, which means obtaining the same results as FedAvg with less server communication, or (ii) for the same communication cost FedDec can converge faster than FedAvg [47].

Furthermore, our analysis reveals that the term improved in the bound is multiplied by a constant α that depends on the spectrum of the inter-agent communication graph. This constant vanishes quickly the more connected the network is, and therefore, the more connected the network the smaller is the impact of the server communication round period H. While a few works have considered inter-agent communication within FL in the past [57–61], none of them has characterized analytically how inter-agent communication reduces the impact of the server communication period H on convergence, and in particular, how this reduction depends on the inter-agent connectivity.

Our simulations confirm our theoretical findings: in all cases, FedDec converges faster than FedAvg, and the gains are greater as either H or the connectivity of the network increase. These results indicate that the reallocation of the resources enabled by inter-agent communication in FL could allow for reaching the target accuracy in less wall-clock time.

We conclude this chapter considering whether the server is even necessary in settings where inter-agent communication is possible, and conversely, how often communication rounds with the server should be if the inter-agent communication graph is not connected (case in which the server becomes indispensable). These and other trade-offs in peer-to-peer communication within FL are interesting directions for future work.

The contributions of this chapter appear in:

• Costantini, M., Neglia, G., & Spyropoulos, T. "FedDec: Peer-to-peer Aided Federated Learning," submitted to *2023 IEEE 62st Conference on Decision and Control (CDC).*

1.1.4 Chapter 5: conclusion

We review the methods proposed in each chapter and their contributions to achieving better utilization of the network resources and higher system performance. We recall the trade-offs involved in each case and the scenarios where each method is expected to provide significant gains. We conclude by enumerating a few orthogonal approaches in the literature that could be combined with the methods presented here to further improve performance.

Chapter 2

Joint design of caching and recommendations

2.1 Introduction

Storing popular items at the edge of the wireless network (i.e. *caching*) is one of the main solutions that have been proposed to deal with the large, ever-increasing demand of content stored in the cloud [62, 63]. Caching at the network edge is beneficial for both the users, who can get content at a higher bitrate and reduced latency [64, 65], and the network operators, who can thus alleviate the congestion during peak traffic periods by having a large fraction of requests being served by the local caches [66]. The cached content can then be updated during off-peak hours.

Recommendation systems, on the other hand, are engines embedded in content distribution applications (YouTube, Netflix, Spotify, etc.) trained to offer the user contents that with high probability will interest them. Recommendation systems are known to greatly affect user requests: it is reported that up to 80% of requests on popular content distribution platforms come from user recommendations [17, 18]. Nevertheless, the traditional role of a recommendation system has been to bring forward items from a vast catalog that best match the users' interests. Where the content is cached and how this affects delivery (e.g., streaming rate) does not usually play a role in the decision of most recommenders.

Recently, major content providers like Netflix and Google started partnering with Internet service providers to implement their own content distribution network solutions inside the network [15, 16]. This naturally brings together content caching and recommendations, as now the same entity can control and coordinate both, towards better user experience, lower network costs, or both. As a result, some recent works propose caching and recommendation policies that take into account this interplay [19, 67–72]. Nevertheless, many of these works still focus on one side of the problem, e.g., network-friendly recommendations [68], [72], or recommendation-aware caching policies [19]. Some works that do try to modify both the caching and recommendation policies are usually based on heuristics [69], [71].

In this chapter we consider the design and analysis of caching and recommendation policies under the *soft cache hits* model [19]. This model assumes that, if the requested content is not locally cached, the user might agree to accept an alternative (but related) content that *is* locally available. One reason for this acceptance might be better streaming quality for similarly interesting content (e.g., it is known that low bitrate can lead to increase in the abandonment rate [73]). Another reason is that the network or content provider that benefits thus from network cost reduction, is willing to offer appropriate incentives to counterbalance the occasional utility loss (e.g. zero-rating). While soft cache hits were formally introduced in [19, 74], other related works also implicitly or explicitly assume soft cache hits. E.g, the pliable index coding of [72] allows for content of lower preference to be served to the user, if that benefits the transmission scheme, while the works of [69, 70] introduce a "distortion" parameter that allows lower (but bounded) relevance content to be recommended if that content is cached. Hence, the framework of soft cache hits targetted in this work has more general applicability.

The first main contribution of this chapter (Section 2.3) is to propose departing from the assumption of [19] that any content in the cache can be recommended to the user. This new setting is significantly more realistic, since standard caches at the edge can contain in the order of 100-1000 contents [15], while only a few 5-20 contents may be recommended to the user in small devices such as smartphones or laptops. However, we now need to choose not only the contents to cache but also the recommendations offered to the user, which renders the problem considerably more challenging. Our contributions in this respect are:

1. Starting from the framework of [19], we introduce (per user) recommendation variables, and formulate the problem of jointly optimizing both sets of (caching and recommendations) variables.
2. We show that even the simplest version of the joint problem (one user, one cache) is NP-hard, and that straightforward application of existing submodularity-based frameworks [10, 19] does not lead to approximation guarantees.
3. We show that using a primal decomposition of the joint problem into an inner (recommendation related) problem and an outer (caching related) problem, leads to an iterative yet efficient (i.e., polynomial) algorithm with constant approximation to the jointly optimal solution.
4. Using a range of real-world datasets we show that the proposed algorithm always outperforms existing heuristics, and discuss the implications of different content types on relative and absolute performance of these schemes.

The second main contribution of this chapter is to study how the intrinsic relations between contents, which make some of them good substitutes of some others, affect the performance of joint caching and recommendation policies. The idea of *related content* naturally motivates the modeling of the set of potentially requested contents (the *catalog*) as a graph where each content is a node and related contents are linked by a (possibly weighted) edge. The structure of this graph can have a significant impact on the performance of policies for related content delivery: if e.g. there exist tight communities of many interconnected contents, any of these could be recommended as an alternative for

any of the other, potentially increasing the gains of joint recommendation and caching algorithms. The particular contributions here are:

1. We formalize the model of a content catalog as a graph and identify key content graph and network setup parameters[1] (e.g. degree distribution and clustering coefficient for the former, and cache size and number of recommendations for the latter) that have an impact on the performance of the caching and recommendation algorithms. We isolate their individual effect by running experiments with synthetic data and changing one parameter at a time, and observe their impact on the absolute and relative performances of the two policies considered.

2. We validate our findings by running the policies on the real-world datasets introduced before, and attempt an analysis of the outcome based on the graph features of each dataset. Furthermore, we show that modifications of the network setup parameters affect the joint policy's performance in each trace differently, and these differences highly depend on the traces' graph structure.

3. Inspired by the dependencies observed between the graph structural parameters and policy performance, we do a preliminary test on training a classifier to predict whether the relative performance between the policies will exceed a given threshold using only a handful of content graph and network setup parameters as predictor variables, with very encouraging results.

2.2 System setup

We tackle the problem of jointly designing which contents to cache at each small cell (SC) and what contents to recommend to each user in the network. The goal is to maximize the number of requests served with the local caches either by (i) providing the user with their original request ("direct" cache hit) or (ii) offering them an alternative content related to their request and stored locally ("soft" cache hit). Our model relies on the hypothesis that since most of the material distributed by content providers such as Netflix and YouTube is entertainment-oriented, the user can be flexible and may accept (with a certain probability) a substitute that is close enough to their original request. Indeed, recommendations have shown to have a significant impact on user behavior when browsing entertainment-oriented platforms [75]. Therefore, our model seeks to provide the user with good recommendations not only from the point of view of showing relevant content that might interest them, but also that can be retrieved easily to have it fast and in good quality. We describe each of the components of our system model in detail below.

[1]We remark that we will use the word "network" to refer to the communication and content distribution network, while we will reserve the word "graph" for the relational model of the content catalog.

2.2.1 Network and caching model

We consider a set of SCs $\mathcal{M}$ and a set of users $\mathcal{I}$, each of them connected to at least one SC in $\mathcal{M}$. We indicate whether a user i can retrieve content from a base station m with the binary variables $q_{im} \in \{0, 1\}$. Our model can be easily extended to consider the uncertainty in the user-SC association by making $q_{im} \in [0, 1]$ the probability of finding user i in the range of SC m. Each SC is equipped with a cache memory of capacity C, i.e. that can store up to C contents[2]. We use variables $x_{km} \in \{0, 1\}$ to indicate whether content k is stored in the cache of helper m. A user can only retrieve contents from the SCs they are connected to and only if the cache contains the file.

2.2.2 Content graph model

We denote with $\mathcal{K}$ the catalog of contents from which the users make their requests. Since many pairs of contents in $\mathcal{K}$ may be related to each other, the catalog can be represented by a graph where each content is a node and related contents are linked by a weighted edge indicating the level of relevance of one content with respect to the other. For example, if $\mathcal{K}$ is a catalog of music videos, two songs of the same artist will be linked with a high weight and two songs from different artists but of the same genre will be linked with a low weight. Therefore, the relations between the contents in $\mathcal{K}$ can be represented with an adjacency matrix $U = \{u_{kn} \in [0, 1]\}, k, n = 1, \ldots, K$ indicating the relative value that a content has to replace another. The values are normalized to have $u_{kn} = 1$ if $n = k$. From now on we will refer to this model of the catalog as the *content graph*.

2.2.3 Recommendation model

In our model each user generates random iid requests for contents in catalog $\mathcal{K}$ where content k is requested with a probability p_k that might be different for each user (in which case we will have a p_k^i for each user i). In our simulations, the p_k follow a Zipf distribution, as shown in related literature [78].

When a new request for a content k arrives from a user i, three things can happen:

1. **Direct hit** ($q_{im} = 1, x_{km} = 1$): The content is found in one or more caches that the user is connected to.

2. **Soft hit** ($x_{km} = 0\ \forall m / q_{im} = 1;\ \exists\ n, m : u_{kn} > 0, x_{nm} = 1, q_{im} = 1$): The caches do not contain the requested content k but the user is offered a maximum of N alternative contents n (the recommendations) that are related to the request and found in the caches. If the user accepts to take any of the recommendations instead, a soft hit occurs whose value depends on the u_{kn}.

[2] We assume for simplicity that each content has roughly equal size (e.g., video chunks), as is commonly assumed in related work [10, 19, 76]. However, our method can be applied to variable size content as well, giving rise to 'knapsack-type' constraints [77].

Table 2.1: Important notation

$\mathcal{M}$	Set of SCs ($\lvert\mathcal{M}\rvert = M$)
$\mathcal{I}$	Set of users ($\lvert\mathcal{I}\rvert = I$)
q_{im}	User i is in range of SC m ($q_{im} = 1$) or not ($q_{im} = 0$)
C	Storage capacity of a SC
x_{km}	Content k is stored in SC m ($x_{km} = 1$) or not ($x_{km} = 0$)
$\mathcal{K}$	Set of contents, catalog ($\lvert\mathcal{K}\rvert = K$)
u^i_{kn}	Utility of content n for a user i requesting content k
p_k	Probability of content k being requested
y^i_{kn}	Recommend n when i requests k ($y^i_{kn} = 1$) or not ($y^i_{kn} = 0$)
N	Maximum number of items that can be recommended per request

3. **Cache miss** ($x_{km} = 0, u_{kn} = 0\ \forall i, k, n, m/q_{im} = 1, x_{nm} = 1$): Neither the requested item nor any related content is found in the accessible caches.

Our model assumes that given a request for content k that is not locally available, the user will be sequentially offered alternative contents n that they might accept with a certain probability $v_{kn} \in [0, 1]$. It is reasonable to assume that the probability of accepting a substitute depends directly on how related the substitute is to the content originally requested (and this is what standard recommenders do, e.g. through collaborative filtering). Thus, in the following we will assume $v_{kn} = u_{kn}$, and discard the notation v_{kn}. This model for soft cache hits was first introduced in [19]. However, here we make the assumption that the number of contents accessible by the user is limited by the number of recommendations. While in [19] it was assumed that *any* content stored in the cache could be offered to the user as a substitute of their request, this is actually unrealistic, since the number of contents in the cache may be much larger than what is practical to recommend through an application interface. This new assumption, however, introduces an additional variable to optimize over (the y_{kn}) and an additional constraint ($\Sigma_n y_{kn} \leq N$).

We also consider the possibility that even if two contents k and n are related, the utility of getting one of them when the other is requested depends on the user's particular preferences and thus it may differ between users. Thus in the multi-user case we will consider individual matrices $U^i = \{u^i_{kn}\}$ for each user i. Since the utility of each content has a direct impact on whether it is recommended or not, we will define the user-specific recommendation matrices $Y^i = \{y^i_{kn}\}$ analogously.

Table 2.1 summarizes the notation of the variables described above.

2.3 Joint problem formulation and joint caching and recommendation algorithm

Based on the previous setup description, it is clear that there are two sets of variables to optimize: what is cached where (variables x_{km}) and what to recommend to each user (variables y^i_{kn}). We formalize our objective below.

2.3.1 Joint optimization: single-user single-cache

In order to better illustrate the problem's hardness, our methodology, and the intuition behind it, we will first tackle the problem of designing the caching and recommendation variables in the single-user single-cache scenario, and we will then extend this result to the multi-user femto-caching setting [79]. We will evaluate the performance of each policy by its attained Cache Hit Ratio (CHR), which measures the expected quotient of the number of hits to the number of requests.

Optimization Problem 1 (Joint Caching and Recommendation - Single User, Single Cache)**.**

$$\underset{x,Y}{\text{maximize}} \quad \text{CHR} = \sum_{k=1}^{K} p_k \left[1 - \prod_{n=1}^{K} (1 - x_n \cdot u_{kn} \cdot y_{kn}) \right] \tag{2.1}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} x_k \leq C \tag{2.2}$$

$$\sum_{n=1}^{K} y_{kn} \leq N, \forall k \tag{2.3}$$

$$x_n, y_{kn} \in \{0, 1\} \tag{2.4}$$

In the problem above the objective function (2.1) computes the expectation of a cache hit over all contents in the catalog. A (potentially soft) hit occurs when a content k is requested and a related ($u_{kn} > 0$) content n that is cached ($x_n = 1$) is recommended ($y_{kn} = 1$). In case of a direct hit, we assume that $u_{kk} = 1$ for all k, and we set $y_{kk} = 1$, i.e. a requested content is always recommended if it is cached (and the user accepts it with probability 1, since they got what they wanted). It is easy to see that the product term in (2.1) corresponds to the probability that no content n leads to a cache hit: this is clear if the content is not cached ($x_n = 0$), or it is not recommended ($y_{kn} = 0$); if it is cached and recommended, then the user rejects it with probability $1 - u_{kn}$. Constraint (2.2) states that we cannot cache more contents than the capacity of the cache, and constraint (2.3) limits the number of recommended items to N. Altogether, our general objective could be loosely described as "maximizing the cache hit rate for the operator, while at the same time making sure that the recommendations to each user are relevant".

Previous works considering the possibility of enhancing the cache hits through recommendations [19, 69, 70] solve a different or partial version of this problem. In [69, 70] the model captures the "distortion" by alternative recommendations in a different manner

(as a constraint outside the objective), and the algorithm solves the problem for x_n only and amends the recommendations in a posterior step. In [19] only the caching problem (choosing x_n) is solved without accounting for the selection of the limited number of recommendations to show to the user. Problem 1 can be reduced to that of [19] by setting $N = C$ and making $y_{kn} = 1\ \forall k, n$.

Note that the above problem is already NP-hard, even for a single cache and one user, since the simpler problem in [19] considering variables x_n only is already NP-hard. However, the objective can be decomposed to optimize over each variable separately and iteratively, which allows to define a polynomial-time algorithm with approximation guarantees. We remark that this is not always the case, and submodularity methods such as those in [10, 19] cannot always be extended directly on both sets of variables (this is easy to see with a counterexample and was proved formally in [70]).

Our method solves a primal decomposition of the joint problem in (2.1): An *outer problem* that maximizes its objective with respect to x and an *inner problem* that, for a given x, maximizes the objective in (2.1) with respect to y. This decomposition is equivalent to the original problem and allows to define a "nested" algorithm where the outer loop tries to find the best content to add to the cache, and the inner loop selects the best recommendations for the user for each potential cache configuration. We provide the details in the following.

2.3.2 Approximation algorithm for problem 1

Let us first assume that the caching vector (variables x_n) are given and consider the subproblem of maximizing (2.1) with respect to variables y_{kn}. We will show first that this can be done in polynomial time, and it will constitute a subroutine of our algorithm.

Let us denote the terms in the objective as

$$f_k(x, Y) = 1 - \Pi_n \left(1 - x_n \cdot u_{kn} \cdot y_{kn}\right). \tag{2.5}$$

Let us further denote as

$$f_k^*(x) = \max_{\substack{Y \\ s.t.\ (2.3),\ (2.4)}} \{1 - \Pi_n \left(1 - x_n \cdot u_{kn} \cdot y_{kn}\right)\}. \tag{2.6}$$

Lemma 1. *Let*

$$F^*(x) = \max_{\substack{Y \\ s.t.\ (2.3),\ (2.4)}} \{\Sigma_k p_k \left[1 - \Pi_n \left(1 - x_n \cdot u_{kn} \cdot y_{kn}\right)\right]\},$$

then $F^*(x) = \sum_k p_k \cdot f_k^*(x)$.

Proof. This follows easily from the fact that the constraints for Y (i.e. Eq. (2.3)) are decoupled per line, so finding the optimal recommendations, given a caching vector, decouples to K independent subproblems of maximizing one term in the sum (corresponding to a row k of Y) subject to the respective constraint (3) for line k. □

Hence, we can focus on optimizing each term in the sum (corresponding to each line k of matrix Y) separately.

Given this observation, Lemma 2 gives the optimal choice of recommendation variables for each possible caching configuration. We denote $S = \{j \in 1, K : x_j = 1\}$ the set of cached elements, and use notation $f_k^*(S)$ and $f_k^*(x)$, interchangeably.

Lemma 2. *Let $u_k^{(j)}(S)$ denote the j^{th} order statistic of elements $\{u_{kn} : n \in S\}$, i.e., the j^{th} largest element in that set. Then,*

$$f_k^*(S) = 1 - (1 - u_k^{(1)}(S)) \cdot (1 - u_k^{(2)}(S)) \dots (1 - u_k^{(N)}(S)). \tag{2.7}$$

Proof. We need to choose at most N contents to recommend, so as to maximize $1 - \Pi_{n \in S}(1 - u_{kn} \cdot y_{kn})$. Assume that choosing the N cached elements with the highest u_{kj} values for each potential request k was not optimal, as claimed above, and recommending another element $l \in S$ leads to a better objective. Let us further assume that l replaces the N^{th} highest content. Then, our assumption implies

$$\begin{aligned} 1 - (1 - u_k^{(1)}(S)) \dots (1 - u_k^{(N-1)}(S)) \cdot (1 - u_{kl}) > 1 - (1 - u_k^{(1)}(S)) \dots (1 - u^{(N)}(S)) \\ \Rightarrow (1 - u^{(N)}(S)) > (1 - u_{kl}) \\ \Rightarrow u^{(N)}(S) < u_{kl} \end{aligned}$$

which is clearly a contradiction, since we assumed that l is *not* among the N-highest u_{kj} values in the cached elements set S.

It is easy to see that the proof holds (in fact strengthens) when replacing any other order statistic apart from the N^{th} in Eq.(2.7). □

The following lemma and theorem are used to prove Corollary 1 below, which will be essential to define our polynomial-time algorithm. The proofs of the lemmas are given in the appendix.

Lemma 3. *$f_k^*(S)$ is a monotonically non-decreasing function of $|S|$, the cardinality of set S.*

Proof. The proof can be found in Appendix A.1. □

Theorem 4. *The function $f_k^*(S)$ is submodular in S, for any k.*

Proof. The proof can be found in Appendix A.2. □

Corollary 1. *The objective of Problem 1, i.e. eq. (2.1), is monotone submodular in x.*

Proof. The objective of Problem 1 is equivalent to (Lemma 1)

$$\underset{x}{\text{maximize}}\ F^*(x) \equiv \underset{x}{\text{maximize}} \sum_k p_k \cdot f_k^*(x)$$

since it holds that [80]:

$$\max_{X,Y} g(X,Y) = \max_{X}\Big(\max_{Y} g(X,Y)\Big)$$

Therefore the objective is a positive weighted sum of monotone submodular functions $f_k^*(S)$ (Lemma 3 and Theorem 4), and hence the sum is monotone submodular as well [77]. □

The submodularity and monotonicity properties of the objective function of Problem 1 allow us to define a primal decomposition algorithm with performance guarantees. We propose Algorithm 1 to design the cache vector x and recommendation matrix Y defined in the problem.

The Algorithm works as follows: in an outer loop (lines 6-15) it looks for the content j that maximizes the marginal gain obtained from adding that content to the cache, as is standard with greedy algorithms for submodular problems. However, for each candidate content it performs an inner loop that sorts the utility values for the new cache configuration and chooses the recommendations in accordance with Lemma 2 (function ChooseRecommendations() in line 11), i.e. for each content k it selects the N cached items with highest u_{kn} and sets $y_{kn} = 1$. The procedure is then repeated until the cache is full.

Algorithm 1 Joint caching and recommendation algorithm (JCR)

1: **Input:** U, N, C, p
2: **Output:** x, Y
3: $x = 0_K, Y = 0_{K\times K}$
4: **while** $t < C$ **do**
5: $x^{\text{aux}} = x, R = 0_K$
6: **for** $j = 1, \ldots, K$ **do**
7: **if** $x_j == 1$**:** ▷ If content j is already cached
8: **continue**
9: **else:**
10: $x_j^{\text{aux}} = 1$ ▷ Try what happens if content j is added to the cache
11: Y = ChooseRecommendations(x^{aux}, N, U)
12: $R_j = \Sigma_k p_k(1 - \Pi_n(1 - x_n^{\text{aux}} \cdot y_{kn} \cdot u_{kn}))$ ▷ Compute hit ratio for this cache configuration
13: $x_j^{\text{aux}} = 0$ ▷ Return cache to its initial state
14: **end if**
15: **end for**
16: $j^* = \text{argmax}_j R_j$
17: $x_{j^*} = 1$
18: **end while**
19: Y = ChooseRecommendations(x, N, U)
20: **return** x, Y

Theorem 5 states the approximation guarantees of Algorithm 1 for the Joint Caching and Recommendation problem for a single cache:

Theorem 5. *Let x_{OPT} be the optimal caching vector to Problem 1 and x^* the caching vector returned by Algorithm 1.*[3] *It holds that*

$$F^*(x^*) > \left(1 - \frac{1}{e}\right) F^*(x_{OPT})$$

Proof. As stated by Corollary 1, the objective function of Problem 1 is monotone submodular, and the problem has a cardinality constraint. It is known that for this class of problems the greedy algorithm achieves in the worst case a $(1 - \frac{1}{e})$-approximation solution [77]. □

2.3.3 Joint optimization: multi-user multi-cache

The results obtained above for a single cache can be extended to prove submodularity and monotonicity also in the multi-user femto-caching (i.e. multi-cache) scenario. In this new setting, a user $i \in \mathcal{I}$ may be connected to one or more helpers $m \in \mathcal{M}$. We will indicate this association with the indicator matrix q_{im}. We also consider now that the utility of a content to replace another may be user-dependent, and thus now we have a per-user content relation matrix $U^i = \{u^i_{kn}\}$ and an associated recommendation matrix $Y^i = \{y^i_{kn}\}$. We will use $\mathbf{U}$ and $\mathbf{Y}$ to denote the 3-dimensional arrays resulting from the concatenation of the U^i and Y^i matrices, respectively. The problem of maximizing the cache hit ratio with soft cache hits and limited number of recommendations in this new scenario is stated in Problem 2.

Optimization Problem 2 (Joint Femto-caching and Recommendation - Multi User, Multi Cache, and User-specific Recommendations)**.**

$$\underset{X,\mathbf{Y}}{\text{maximize}} \sum_{k=1}^{K}\sum_{i=1}^{U} p^i_k \left[1 - \prod_{n=1}^{K}\left[\prod_{m=1}^{M}(1 - x_{nm}\cdot q_{im}) + \left(1 - \prod_{m=1}^{M}(1 - x_{nm}\cdot q_{im})\right)(1 - u^i_{kn}\cdot y^i_{kn})\right]\right] \tag{2.8}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} x_{km} \leq C, \qquad \forall m \in \mathcal{M} \tag{2.9}$$

$$\sum_{k=1}^{K} y^i_{kn} \leq N, \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{U} \tag{2.10}$$

$$x_{nm}, y^i_{kn} \in \{0, 1\} \tag{2.11}$$

The steps in the proof of Theorem 4 allow us to easily extend the submodularity and monotonicity results to the multi-user femtocaching scenario of Problem 2, as shown next.

Corollary 2. *The problem of joint femto-caching (i.e. multiple co-dependent caches) and recommendation is also submodular (subject to matroid constraint).*

[3]Note that the optimal recommendation matrices Y_{OPT} and Y^* are immediately obtained from x_{OPT} and x^* respectively by recommending the N cached elements with the highest utility (see Lemma 2)

Proof. The proof can be found in appendix A.3. □

A greedy algorithm analogous to Algorithm 1 can be defined for Problem 2, where now the outer loop goes through all potential (content, cache) assignments (see Appendix A.3). This algorithm can guarantee a $\frac{1}{2}$-approximation of the optimal solution, since Problem 2 is a maximization problem with a submodular objective and a matroid constraint [81].

2.4 Performance of JCR versus independent cache and recommendation design

We performed simulations with both synthetic and real-world data to test the performance of our algorithm, JCR, which designs the caching and the recommendation jointly, against two alternative approaches that design these variables separately, introduced next.

Baseline caching policies:

1. Soft Cache Hits (SCH): Chooses the contents to store taking into account the possibility of having soft cache hits but *without* considering the limited number of recommendations. This method is identical to that in [19] and implicitly assumes that all cached contents can be recommended.

2. Popularity caching (POP): Caches the most popular contents until the cache is full. This approach is completely blind to the possibility of satisfying the user request with alternative but related content.

Recommendation policy: The recommendations in the two approaches above are chosen *after* having filled the cache with their respective criteria, and these are chosen in accordance with Lemma 2 (for each content k, the N contents with highest u_{kn} are recommended).

For our implementation of the greedy-based approaches (JCR and SCH) we used the *lazy evaluations method*, which exploits the submodularity property of the objective to significantly accelerate the greedy search process [77].

2.4.1 Synthetic data

We performed experiments with synthetic data to test the effect of the graph structure and the network parameters on the performance of the algorithms. This allowed us to appreciate the relative performance of the schemes in a controlled environment for different setup parameters before testing them on the real-world data. Figure 2.1 shows these results for a synthetic content graph generated with the Barabassi-Albert model and two different scenarios: (1) catalog size K=500, cache size C=50 and number of links added by each new node n=10, and (2) K=1000, C=75 and n=2. In both cases the recommendations were N=1, the popularity Zipf exponent $\alpha = 1$, and the u_{kn}=0.5.

Figure 2.1 shows that JCR can beat the other two algorithms in both scenarios and achieve over a 10% of difference with the weakest, but there is a trade-off that (with this particular graph structure) does not allow it to significantly beat both at once. In Scenario 1 the high C/K ratio and the large number of edges favor the "confusion" of SCH, which adopts the strategy of trying to accumulate many soft hits but in the end its performance is constrained by the limited number of recommendations. On the contrary, POP can accrue a lot of direct hits thanks to the large C, and its CHR is further boosted by the soft hits obtained thanks to the high connectivity of the graph. The situation is reversed in Scenario 2: low connectivity and low C/K ratio harm the performance of POP but push SCH towards adopting a strategy more similar to that of JCR, where the fractions of soft and direct hits are more balanced than in Scenario 1. The differences between JCR and SCH will be more notorious when $N \ll C \ll K$, as in real scenarios. However, to appreciate them in our limited-size datasets we had to set it to $N = 1$. As can be expected, increasing N with C fixed makes SCH perform more similarly to JCR, and identically in the limit $N = C$.

In Section 2.5 we go deeper into the impact of these system choices on the performance of JCR and POP, taking into account also the role of the graph properties of the content graph.

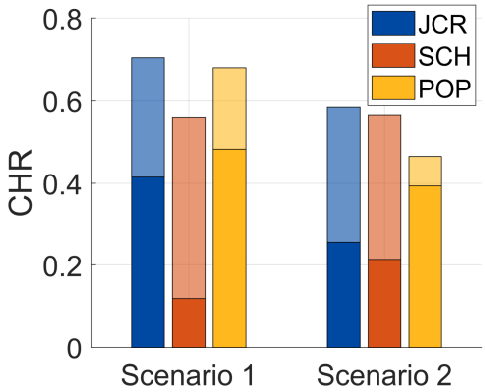

Figure 2.1: Expected cache hit ratio (CHR) of the three algorithms tested in the two synthetic scenarios. The fraction of CHR earned by direct and soft hits have both been indicated in each bar in dark and light colors, respectively.

2.4.2 Real-world data

For our experiments with real data we considered four datasets:

MovieLens (K=3306): for this dataset we built $\mathbf{U}$ from the user ratings using collaborative filtering (see details of preprocessing in [19]).

YouTube (K=2098) [82]: here we built matrix $\mathbf{U}$ setting u_{kn} to non-zero if k is in the list of recommended videos for n or vice versa.

We also considered two datasets not related to video content to further compare the algorithms:

Amazon Videogames (Azn-VG, K=5614) and **Amazon for Android applications** (Azn-Apps, K=8229) [83]: For these we built $\mathbf{U}$ from the "also bought" list of the datasets, setting $u_{kn} \neq 0$ when items k and n were bought together.

In all cases we kept the largest component of the dataset graphs, and we set the off-diagonal elements of U to $u_{kn} = 0.5$. Since the Amazon datasets did not contain the content popularity distribution, we synthetically generated random popularity values for each content following a Zipf distribution with exponent $\alpha = 2$. In our experiments we set the cache size C to be 5% the size of the dataset catalog and $N = 1$ recommendation was shown to the user for each content request.

Figure 2.2 shows the performance of our algorithms in the single-cache scenario for all datasets. We have indicated the fraction of cache hits coming from direct and soft hits in dark and light colors, respectively. Below we make some remarks on these results.

The joint approach outperforms the other two in all cases. In all cases JCR achieves a higher CHR that the other two approaches (both in real and synthetic datasets). This demonstrates the importance of not only accounting for the possibility of soft cache hits, but also for the limited number of recommendations. It is worth noticing that even if greedy has a worst case approximation in theory, in practice it performs very close to optimal.

The magnitude of the difference between JCR and the other algorithms depends very much on the source of the greatest fraction of hits (direct or soft). Whether JCR can beat one or both of the other algorithms by a large difference depends on *how* it does so. In the YouTube dataset, for example, all approaches perform similarly and they go for a lot of direct hits. This suggests that for this dataset the gains are dominated by the popularity values and even JCR and SCH tend to adopt the strategy of POP. Conversely, for MovieLens it seems that the graph structure benefits the acquisition of soft hits. Thus in this case the approaches accounting for soft hits (JCR and SCH) prioritize this source of CHR and adopt a similar strategy that significantly outperforms POP. The greatest gains of JCR with respect to the other two are better appreciated when similar gains can be obtained either by caching popular items or by exploiting the soft hits, as observed in the Azn-VG and Azn-Apps datasets.

The overall performance of each algorithm and the differences between them highly depend on the graph structure. As shown in the experiment with synthetic data, by just modifying the graph properties we can invert "who JCR beats by a large difference". The real traces show even more variability in the performance of the three algorithms, both in terms of their ranking order and the source of the largest CHR fraction achieved by each of them (if either by direct or soft hits).

Motivated by the last observation, in the next section, we investigate more deeply (i) the interplay between the system parameters (values C and N) and the structure of the content graph, and (ii) their impact on caching and recommendation policies.

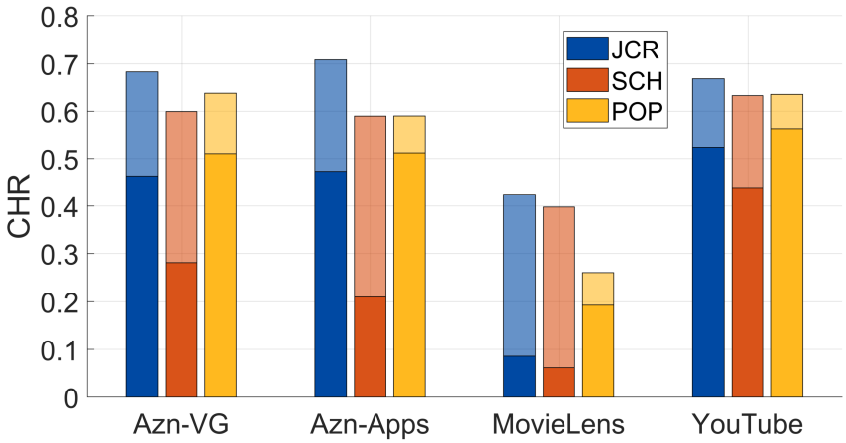

Figure 2.2: Expected CHR of the three tested algorithms in all four datasets.

2.5 Impact of the content graph structure and the system setup

In Sections 2.3 we presented JCR, a polynomial-time algorithm with approximation guarantees for solving the (NP-hard) problem of designing caching and recommendations jointly. However, in large instances, even polynomial-time approximation algorithms can become prohibitively slow. Furthermore, such algorithms provide (close to) optimal values of the control variables, but no insights as to what key parameters lead to these choices, or why the gains are sometimes moderate and others large, depending on the catalog considered. This motivates us to ask a number of questions:

1. What is the added value of doing the joint optimization with respect to a fast heuristic (e.g. POP) when considering e.g. the Netflix catalog of movies?

2. How does this value depend on key properties of the graph? Will it change if we consider a catalog of Amazon Prime movies instead?

3. Could we use these properties to predict performance on a new catalog, without actually running the optimization algorithm?

Here we carry out a preliminary investigation of these questions, by identifying *which* characteristics of the content graph affect the policies for caching and recommendation and *how* they do so. To better illustrate our methodology and findings, we will consider only the JCR and POP policies in this section, since they represent two extremes of complexity: JCR needs to run Algorithm 1, while POP simply caches the most popular files and recommends the most relevant cached items as proposed in Lemma 2.

This analysis, as we show in Section 2.7, may allow us to state *a priori* whether, for a given content graph, it is worth finding the optimal (or close to optimal) cache configuration and content recommendations, as we do with JCR, or a low-complexity heuristic such as POP will perform similarly enough, only based on the graph features and without having to actually run the complex algorithm.

Table 2.2: Graph parameter notation

R	Node degree
$E[R]/K$	Density
α	Zipf popularity exponent
p_{link}	Link probability in an ER graph
ℓ_{new}	Number of links added per node in a BA graph
$\gamma(X)$	Skewness of random variable X
κ	Clustering coefficient
n	Newman's modularity

2.6 Numerical tests of parameter impact

In this section we concentrate on a few aspects of the content graph structure and study their effect on the performance of the policies introduced before. We also consider different system parameters (e.g. cache sizes) which, as we will show, can further affect the (absolute and relative) impact of the content graph properties.

Understanding the impact of these parameters can be helpful for, given a content graph of interest, deciding whether it is worth running a computationally expensive but close-to-optimal algorithm over a quick and simple heuristic (see Section 2.7 on performance prediction), for designing the network setup to make the most of a particular policy (see Fig. 2.5 on the impact of N and C on different real-world traces), or to define the limits of performance achievable by any approach.

A fair question to ask at this point is *how do we characterize the structure of a graph?* The answer is not straightforward, since graphs are complex structures over which many metrics can be computed (see e.g. [84]). Furthermore, an exhaustive characterization (through the adjacency matrix, for example) is not insightful and impractical for large graphs. Here we have concentrated on a few macroscopic parameters that can be easily measured in any graph and that we could expect to have a significant impact on the performance of the joint caching and recommendation problem.

In the first part of this section, we perform experiments with synthetic data changing one parameter at a time to isolate their effect on the policies' performance as much as possible. In the second part, we measure the graph parameter values of real-world traces and look at the performance of the two considered policies on them. We then attempt to interpret the latter results in light of the trends identified with the synthetic data.

Table 2.2 summarizes the notation used in this section, where the skewness $\gamma(X)$ of a random variable X is measured with Pearson's moment coefficient $\gamma(X) = E[((X - \mu_X)/\sigma_X)^3]$, and the clustering coefficient κ is measured as the number of triangles over the number of triples in the graph.

Table 2.3: Parameter values used in each experiment with synthetic data. Each column shows a different test, and the cell of the variable tested is highlighted in gray.

	Experiments				
Feature tested	Degree skewness	C	Zipf exponent	N	Community structure
Graph	{ER, BA}	ER	ER	ER	{ER, comm}
K	1000	1000	1000	1000	1000
C	10	{10, 50, 100}	50	50	50
N	1	1	1	{1, 2, 3}	1
α	1	1	{0.5, 1, 1.5}	1	1
p_{link}	0.04	0.01	0.01	0.04	0.02
ℓ_{new}	10	-	-	-	-

2.6.1 Synthetic data

We tested the effect of both (i) content graph parameters: degree skewness, popularity skewness and community structure, and (ii) network setup parameters: cache size and number of recommendations. For our simulations with synthetic data we used three probabilistic models of graphs:

• **Erdős–Rényi (ER):** a link between each pair of nodes is generated at random and independently with probability p_{link}. The resulting node degree is a random variable with binomial distribution, i.e. $R \sim B(K, p_{link})$.

• **Barabási–Albert (BA):** starting from an initial graph of K_0 nodes, a new node is added by connecting it with $\ell_{new} \leq K_0$ new links to the nodes already in the graph. The probability of connecting to a node in the graph is proportional to its degree. The degree distribution of the resulting graph follows a power law distribution with exponent 3 [84].

• **Community graph:** this is a disconnected graph where each community is connected and there are no links between different communities.

Next we describe the experiments done to test the impact of each graph parameter on the policies' performance and discuss the results. The parameter values used in each experiment are shown in Table 2.3, where each column indicates a different experiment where the effect of only one feature was tested. The cells showing the different values that the feature of interest takes are highlighted in gray. In all cases and without loss of generality, we set $u_{ij} = 0.5$ if contents $i \neq j$ are related.[4] For all graphs, we assume a Zipf popularity distribution with tunable exponent α, and assign the popularities p_i to the graph nodes randomly. For each experiment we perform 32 repetitions of the graph generation and policy testing. We then report the average and standard deviation of the CHR accrued by each policy.

Degree skewness. The first panel of Figure 2.3 shows the performance of the two policies for both BA and ER graphs with $\ell_{new} = 10$ and $p_{link} = 0.04$, respectively. These

[4]We observed empirically that the choice of this value (barred from the two extremes of 0 and 1) only affects the total CHR but not the qualitative impact of different graph parameters.

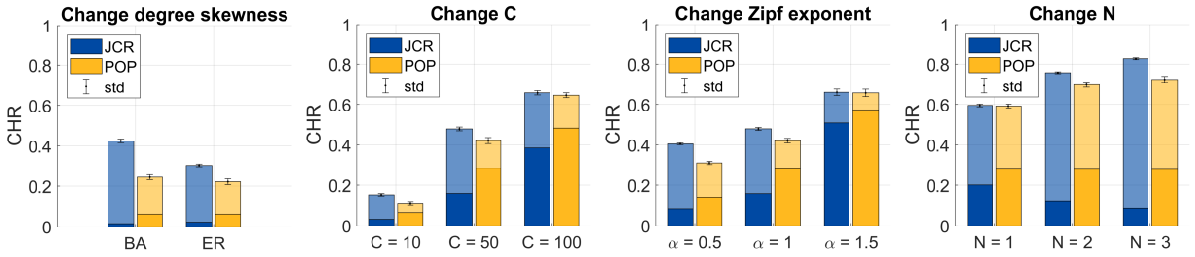

Figure 2.3: Effect of degree skewness, cache size, popularity Zipf exponent and number of recommendations on performance.

values were selected so that the density $E[R]/K$ would be the same for both types of graphs, since a higher density would automatically increase the CHR thanks to more soft hits, and the comparison in that case would be unfair. The bars show the total CHR accrued, and the fraction obtained from direct and soft hits are shown in dark and light color respectively.

The CHR achieved by JCR on the BA graph is much larger than that of the ER graph. This happens because JCR is capable of picking the high-degree nodes in the BA graph to earn many soft hits. The ER graph, however, does not provide such a possibility, since all nodes have approximately the same degree and there is no clearly winning strategy that JCR can take. POP, on the other hand, performs almost identically for both graph types. This is because the popularity values were assigned to the nodes randomly, thus caching the most popular nodes is equivalent to picking up randomly and uniformly C nodes in the graph. Therefore the performance of POP depends on the average connectivity, and since $E[R]/K$ is the same for both graph types, POP achieves approximately the same CHR in both cases.

Cache size. The second panel shows the performance of both algorithms on an ER graph and for different cache sizes. As expected, as the cache size increases both algorithms achieve a higher CHR. However, the high popularity skewness helps POP to boost its performance more than it helps JCR: by making the cache size only a 10% of the total catalog size, POP can achieve with its simple criterion a CHR ≈ 0.5 coming from direct hits and a total CHR comparable to that achieved by JCR. Note that if the popularity distribution was uniform ($\alpha = 0$) the amount of direct hits accrued by POP would equal C/K. The effect of the skewed distribution of p_i is discussed next.

Popularity skewness. The third panel shows the performance for different Zipf popularity exponents. Again, increasing this parameter helps both algorithms, but the boost is larger for POP. The reason is analogous to that of increasing C for fixed α: a larger percentage of requests can be served with direct hits, which is what POP goes after by caching the most popular contents. Note that as α increases, JCR tends to "copy" the strategy of POP and go for more direct hits.

Number of recommendations. The last panel shows the performance when we change N. The gap between JCR and POP increases with N in favor of the former. This is due to POP getting more soft hits just by chance, while JCR adjusts its strategy to exploit the larger number of recommendations. Note that the fractions of soft and direct hits of JCR increase and decrease respectively as N gets larger, and added together they achieve

a higher total CHR each time.

Community structure. Figure 2.4 shows the performance for different cache sizes and two types of graphs: a community graph that contains 50 communities of 20 nodes connected in a clique (i.e. a complete graph) and an ER graph with $K = 1000$ and $E[R] = 20$. The strong community structure enhances the performance of JCR with respect to the ER case, and the effect is larger when the cache size matches the number of communities. This is because JCR can recognize that choosing one content per community is the best strategy. When C is smaller than the number of communities, the effectiveness of this strategy is limited by the cache size. When it is larger, JCR has to pick more than one node per community and the gains with respect to ER are more moderate. Again there are no significant effects on POP when the graph type changes, since popularity values are assigned at random and the density of both graphs is the same.

In the experiments with real-world data of the next section we measured the degree of community structure with two parameters: the clustering coefficient κ and Newman's modularity n [85]. For the latter, we used the implementation of [86], which applies the method of [87] to find communities in the graph before computing n.

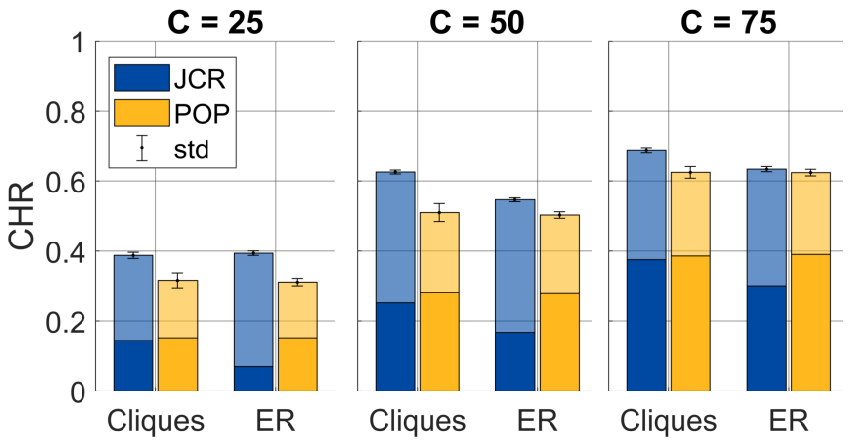

Figure 2.4: Effect of community structure. Community structure boosts the performance of JCR respect to POP, provided that C approximately matches the number of communities.

Overall, the results with synthetic data seem to indicate that there is indeed a direct relation between graph parameters and policy performance. But *can we observe similar tendencies in real-world data?* We address this question next.

2.6.2 Real-world data

We tested the performance of JCR and POP in the datasets AznApp, MovieLens, and YouTube introduced in Section 2.4.2. We also used the LastFM dataset [88], which contains for each song a list of similar songs, and thus we considered songs i and j related when song i was in the list of j or vice-versa.

Like in the synthetic data, we set $u_{ij} = 0.5$. Since the Amazon and LastFM datasets did not contain the content popularity distribution, we generated random popularity values

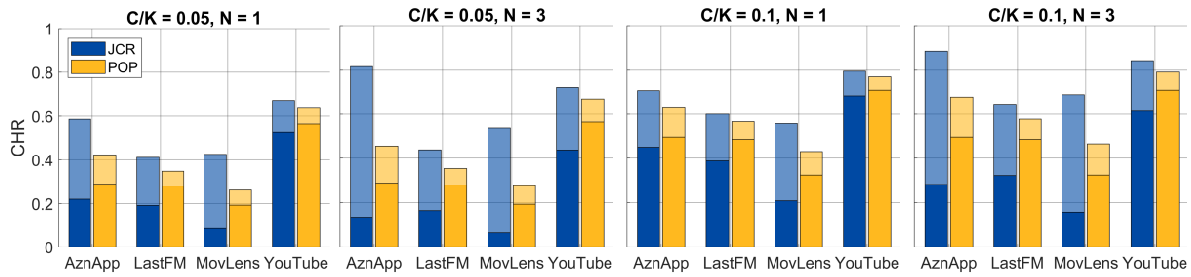

Figure 2.5: Results on traces for different cache sizes and number of recommendations.

following a Zipf distribution with $\alpha = 1$. Table 2.4 shows the graph parameter values of all traces.

Table 2.4: Graph parameter values of the real-world traces. Notation is explained in Tables 2.1 and 2.2.

	AznApp	LastFM	MovLens	YouTube
K	8229	3506	3306	2098
$E[R]$	15.97	4.25	25.49	5.38
$E[R]/K$	0.0019	0.0012	0.0077	0.0026
$\gamma(R)$	11.95	3.6	2.41	1.42
$\gamma(p)$	1.94	1.89	2.07	7.57
κ	0.08	0.13	0.55	0.38
n	0.59	0.81	0.78	0.84

Figure 2.5 shows the performance of both algorithms in the traces for different C/K ratios and values of N. Note that there is no standard deviation specified in these plots, since the traces are taken from real-world data and are thus unique.

As observed in synthetic data, increasing either C or N improves the performance of both algorithms. In the case of POP, increasing C is consistently better than increasing N, as we could expect. For JCR, however, *which* of the two parameters has the greatest impact to boost the performance in a trace *depends heavily on the characteristics of the content graph*. The changes produced by increasing N are seen by comparing the *second* panel with the first one in Fig. 2.5. Similarly, the changes produced by increasing C are seen by comparing the *third* panel with the first one. Our observations are the following:

When the graph structure favors the accrual of soft hits, it is more beneficial to increase N***:*** This is true for AznApp, which has very high degree skewness, thus confirming the observations done with synthetic data. The MovLens dataset has high density and clustering coefficient, which also favors the accrual of soft hits. However, the structure of this trace seems not to be as convenient for JCR as that of AznApp to exploit soft hits, and increasing C achieves a similar total CHR as increasing N.

When the graph is not particularly well-connected or the popularity skewness is high, it is better to increase C***:*** This happens for YouTube, which has a very high popularity skewness, and for LastFM, whose parameters are similar to those of MovLens

but is more sparsely connected (it has significantly lower average dergee $E[R]$, density $E[R]/K$, and clustering coefficient κ). In these cases JCR can do better by accruing direct hits rather than soft ones, and thus increasing C is preferable over increasing N.

These observations suggest that, as observed with synthetic data, qualitative and quantitative performance differences in these much more complex traces can be, to some extent, attributed to structural differences in their content graphs. This motivates our next and final step in this work: to investigate whether these features can be used to predict the expected performance of the policies in a given dataset.

2.7 Performance prediction

The results of the previous section with both synthetic and real-world data show that the performances of JCR and POP are heavily affected by the content graph structure and the network setup parameters. This suggests that looking at the parameters alone might suffice to design an automated performance predictor that, given a new graph (i.e. a new dataset), will be able to decide whether optimizing the caching and recommendations jointly can provide significant benefits over just caching the most popular items without actually having to run the optimization algorithm. Given the limited amount of training data (data catalogs and related content graphs), we choose in this work to perform a simple classification task: *will joint optimization (JCR) provide relative gains that exceed a threshold T, compared to the baseline (POP)?* We will attempt to answer this question using only the graph and network-related features introduced before, and a support vector machine (SVM) classifier.

2.7.1 Dataset

We have chosen SVM because it can achieve reasonable performance even with small amounts of training data (compared to e.g. deep neural networks). Nevertheless, our baseline consists of 6 collected traces, and 2 synthetic graph types. It is thus important to devise a methodology to come up with a proper training set out of these, that has diversity in the graph structure and not too many samples of a particular type of graph that could generate bias. Our complete dataset was constructed from:

Splitting traces: We randomly split the traces in groups of $500 \leq K \leq 700$ nodes (the average number of nodes in our traces was 4257), each of which constitutes a new graph of our dataset. Apart from the four traces used in Section 2.6.2, here we also considered the Amazon Virtual Games (AznVG, $K = 5614$) and Amazon Movies & TV data (AznTV, $K = 2789$) datasets [83]. The adjacency values u_{ij} were obtained as those for AznApp.

Synthetic graphs: To add variability to the dataset and potentially improve generalization we generated six additional cases: three ER graphs with $p_{link} = 0.002, 0.005$ and 0.01 respectively, and three BA graphs with $\ell_{new} = 1$, 4 and 8. The number of nodes of these graphs was chosen randomly and uniformly in [500, 700].

This procedure provided a total of 47 graphs (41 from real traces, 6 from synthetic data). We will refer to the set of graphs generated from a particular real-world dataset or synthetic graph as their "child graph", e.g. out of the YouTube dataset we generated 36 YouTube child graphs with this subsampling method. For every such child graph, we also create different network setups from combinations of the following parameters: (i) 3 values of Zipf popularity $\alpha = 0.5$, 1 and 1.5, (ii) 3 values of $C/K = 0.01$, 0.05 and 0.1, and (iii) $N = 3$. This gives a total of 9 network setups per child graph, and thus a total of 423 different scenarios to consider for training (and testing).

2.7.2 Experiment design

Each case i in the dataset was represented by a vector $x_i \in \mathbb{R}^7$ with the values of the parameters $\{K,\ E[R],\ \gamma(R),\ \kappa,\ n,\ \alpha,\ C\}$. We assigned a binary label y_i to each case according to:

$$y_i = \begin{cases} 1 & \text{if } \frac{\mathrm{CHR}_{\mathrm{JCR}}(i) - \mathrm{CHR}_{\mathrm{POP}}(i)}{\mathrm{CHR}_{\mathrm{JCR}}(i)} > T \\ -1 & \text{otherwise,} \end{cases}$$

where $T \in [0, 1]$ is a threshold for relative performance that we chose arbitrarily, and the subindex of $\mathrm{CHR}_{\mathrm{P}}(i)$ indicates the CHR obtained by policy P when applied to case i. The choice of the threshold T defines the number of cases with each label. Table 2.5 shows the distribution of labels of the child graphs for each real-world or synthetic dataset and the two thresholds $T = 0.1, 0.15$ used in our experiments. Note that the addition of the two numbers gives the total number of child graphs for each dataset.

To test the generalization power of the obtained model, we carried out the training using the child graphs of 7 out of the 8 datasets considered and used the remaining dataset for testing. This resulted in 8 different training and testing experiments, each using a different dataset as testing data. Such a splitting allowed us to make sure that the test set was new unseen data, completely unrelated from that used for training. Furthermore, to test the robustness of the results we repeated each of the 8 experiments 10 times. We report the mean and standard deviation of the training and testing accuracies in each experiment.

2.7.3 Support vector machines

The SVM is a non-probabilistic binary linear classifier method that finds the hyperplane in the feature space that better separates the two classes. The optimization problem solved by SVM can be formulated as

$$\begin{aligned} \min_{\beta,\beta_0,\xi} \quad & \frac{1}{2}\|\beta\|^2 + \delta \sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i,\ i = 1, \ldots, m \\ & \xi_i \geq 0,\ i = 1, \ldots, m \end{aligned}$$

Table 2.5: Distribution of labels per set ($y = 1/y = -1$).

Dataset	$T = 0.1$	$T = 0.15$	**Dataset**	$T = 0.1$	$T = 0.15$
AznVG	44 / 37	38 / 43	MovLens	28 / 17	20 / 24
AznApp	66 / 42	57 / 51	YouTube	13 / 23	10 / 26
AznTV	20 / 25	15 / 30	ER	12 / 15	10 / 17
LastFM	13 / 41	11 / 43	BA	21 / 6	18 / 9

where m is the number of samples in the data (in our case graph-network setup combinations), $f(x) = \beta^T x_i + \beta_0$ defines the separating hypeplane between the two classes, ξ_i are slack variables that allow for having points on the wrong side of the boundary and δ is a cost parameter that penalizes having a large number of misclassified points [89].

For our experiments we normalized the features x_i by subtracting their mean and dividing by their standard deviation. For the training of the SVM we used the Matlab function `fitcsvm` and let the software optimize the hyperparameter δ through the option `OptimizeHyperparameters`.

2.7.4 Results of automatic performance prediction

Table 2.6 shows the mean and standard deviation (between parenthesis) of both the training and testing accuracies over the 10 repetitions of the experiments and for two values of T. The distribution of labels ($y = 1/y = -1$) is shown next to each value of T. Next we make some remarks on these results.

Table 2.6: Accuracy of the SVM classifier for relative performance prediction.

Set used for testing	$T = 0.1$ (217/206)		$T = 0.15$ (180/243)	
	Test	**Train**	**Test**	**Train**
AznVG	0.93 (0.006)	0.94 (0.002)	0.96 (0.020)	0.93 (0.003)
AznApp	0.93 (0.012)	0.95 (0.001)	0.89 (0.035)	0.93 (0.005)
AznTV	0.96 (0.000)	0.95 (0.004)	0.86 (0.015)	0.93 (0.002)
LastFM	0.83 (0.020)	0.95 (0.004)	0.86 (0.049)	0.94 (0.007)
MovLens	0.84 (0.007)	0.95 (0.001)	0.91 (0.000)	0.93 (0.003)
YouTube	0.87 (0.037)	0.95 (0.002)	0.94 (0.000)	0.92 (0.003)
ER	0.96 (0.000)	0.95 (0.001)	0.83 (0.031)	0.93 (0.001)
BA	0.91 (0.031)	0.94 (0.001)	0.83 (0.019)	0.93 (0.001)
Mean	**0.90 (0.051)**	**0.95 (0.005)**	**0.89 (0.049)**	**0.93 (0.006)**

The features considered are good indicators to distinguish between high- and low-gain scenarios for the joint approach with respect to the baseline. This observation is based on the high training and testing accuracies observed in all experiments. However, in some cases there is a large gap between the training and

testing accuracies (differences larger than 5% in Table 2.6 have been highlighted in blue), which leaves room for further improvement, discussed below.

More features or non-linear combinations of the current features might be needed when the generalization power changes with T. This comes from the fact that for some traces (AznTV, MovLens, YouTube, ER and BA) the training and testing accuracies are very similar for one value of T and quite different for the other. Thus, for one T the "separation rule" learned from the training data applies well to the test set (i.e. the hyperplane found makes a good split of the testing points), but when the labels change for the new T the rule learned at training does not apply to the test set anymore. For the former case a potential solution is enlarging the training set (discussed in the next point). Another alternative is considering more complex, non-linear interactions between the features: an example of such interplay was the link between the cache size and the number of communities identified in the experiments with synthetic datasets, where the gains of JCR due to strong community structure were maximized when both quantities were equal. Thus, adding new combinations of the features already considered may allow for capturing effects not well represented yet. Adding completely new features could help to this end as well, but contrary to what we want they might broaden the accuracy gap by allowing more easily for overfitting.

Enlarging the dataset with new, different and diverse graphs that enrich the feature space would probably improve generalization significantly. This would most likely help particularly the experiment using the LastFM child graphs for testing, apart from being beneficial also for the cases mentioned in the previous paragraph. When testing with the LastFM child graphs the gap between training and testing accuracies is large for both values of T. This suggests that this test set might have specific characteristics not accounted for in the training set. In such case using a larger dataset with more varied content graphs that are potentially more similar to that of the LastFM dataset could help to improve testing accuracy.

2.8 Conclusion

Caching and recommendation are two distinct technologies that impact each other but that are currently optimized separately. We have proposed a simple algorithm with provable approximation guarantees that can jointly design the caching and recommendation strategies in very general scenarios and consistently outperform schemes that take these choices separately.

Furthermore, we observed how the magnitude of the gains highly depends on the graph structure of the dataset. This motivated us to study how the graph properties of the dataset, the system setup parameters, and the interplay between all of these affect the outcome of the algorithms. For this study, we compared our polynomial-time algorithm against a simple heuristic that caches the most popular contents without taking into account the recommendations. We could associate particular graph properties and setup parameters to specific changes in the outcomes of the two policies considered. Such connection between graph parameters and policy performance reveal that there is a *fun-*

damental limit on the gains that *any* smart policy can achieve, compared to the baseline, that is intricately tied to the properties of the dataset (graph) itself, rather than the algorithm.

These results encouraged us to test whether we could do automatic performance prediction from a few content graph and system setup descriptors. Our results with SVM classification support the hypothesis that with careful training of the classifier and a sufficiently representative dataset it is indeed possible to get good predictions. This not only gives further proof of the tight relation between graph properties and policy performance, but it also opens the possibility of using automatic classification for deciding whether applying a "smart" but computationally costly policy is worthy with respect to just using a simple baseline.

We remark that the JCR algorithm introduced here achieves a higher CHR *without requiring any architectural changes* in the current caching networks and recommendation engines. This means that by using our joint design policy, the network load can be reduced (since fewer contents are fetched from the remote server) without additional costs when it comes to the physical structure of the network. There is, however, the computational cost of running this policy. The performance predictor proposed intends to give a tool to decide whether this cost is worth paying, and thus, in conjunction with the JCR algorithm, they can achieve a better use of the communication *and* the computation resources.

Chapter 3

Local Gauss-Southwell rule for decentralized optimization

3.1 Introduction

Many timely applications require solving optimization problems over a network where nodes can only communicate with their direct neighbors. This may be due to the need of distributing storage and computation loads (e.g. training large machine learning models [20]), or to avoid transferring data that is naturally collected in a decentralized manner, either due to the communication costs or to privacy reasons (e.g. sensor networks [21], edge computing [22], and precision medicine [23]).

Specifically, we consider a setting where the nodes want to solve the decentralized optimization problem

$$\underset{\theta\in\mathbb{R}^d}{\text{minimize}} \quad \sum_{i=1}^{n} f_i(\theta), \tag{3.1}$$

where each local function f_i is known only by node i and nodes can exchange optimization values (parameters, gradients) but *not* the local functions themselves. We represent the communication network as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes (agents) and $E = |\mathcal{E}|$ edges, which are the links used by the nodes to communicate with their neighbors.

Existing methods to solve this problem usually assign to each node a local variable θ_i and execute updates that interleave a local gradient step at the nodes followed by an averaging step (usually carried out by a doubly-stochastic matrix) that aggregates the latest parameter of the node with those of its neighbors [28, 36, 90–93].

Alternatively, the consensus constraint can be stated explicitly between node pairs connected by an edge:

$$\underset{\theta_1,\ldots,\theta_n\in R^d}{\text{minimize}} \quad \sum_{i=1}^{n} f_i(\theta_i) \tag{3.2a}$$

$$\text{s.t.} \qquad \theta_i = \theta_j \quad \forall\ (i,j) \equiv \ell \in \mathcal{E}, \tag{3.2b}$$

where $\ell \equiv (i, j)$ indicates that edge ℓ links nodes i and j.

In this work, we solve the decentralized optimization problem by solving the dual of (3.2), which allows us to propose *asynchronous* decentralized algorithms where each update involves a single pair of nodes and many updates can happen simultaneously. In our setup, nodes activate anytime at random and select one of their neighbors to make an update together. Methods with such minimal coordination requirements avoid incurring extra costs of synchronization that may also slow down convergence, which is the reason why many algorithms for this asynchronous setting have been proposed in the literature [25, 39–42]. However, most of these works assume that when a node activates, it simply selects the neighbor to contact randomly, based on a predefined probability distribution. This approach overlooks the possibility of letting nodes *choose* the neighbor to contact taking into account the optimization landscape at the time of activation. Therefore, here we depart from the probabilistic choice and ask: *can nodes pick the neighbor smartly to make the optimization converge faster?*

In this chapter, we give an affirmative answer and propose algorithms that achieve this by solving the dual problem of (3.2). In the dual formulation, there is one dual variable $\lambda_\ell \in \mathbb{R}^d$ per constraint $\theta_i = \theta_j$, hence each dual variable can be associated with an edge ℓ in the graph. Our algorithms let an activated node i contact a neighbor j so that together they update their shared variable λ_ℓ, $\ell \equiv (i, j)$ with a gradient step. In particular, we propose to select the neighbor j such that the updated λ_ℓ is the one *whose directional gradient for the dual function is the largest*, and thus the one that provides the greatest cost improvement at that iteration.

Interestingly, the above protocol where a node activates and selects a λ_ℓ to update can be seen as applying the coordinate descent (CD) method [43] to solve the dual problem of (3.2), with the following key difference: unlike standard CD methods, where *any* of the coordinates may be updated, now *only a small subset of coordinates are accessible at each step*, which are the coordinates associated with the edges connected to the node activated. Moreover, our proposal of updating the λ_ℓ with the largest gradient is similar to the Gauss-Southwell (GS) rule [45], but applied *only* to the parameters accessible by the activated node.

We name such protocols *set-wise CD* algorithms, and we analyze a number of possibilities for the coordinate (or equivalently, neighbor) choice: random sampling, local GS, and two extensions that take into account the smoothness constants of the dual function. Compared to standard CD literature, three difficulties complicate the analysis and constitute the base of our contributions: (i) for arbitrary graphs, the dual problem of (3.2) has an objective function that is *not* strongly convex, even if the primal functions f_i *are* strongly convex, (ii) the fact that the GS rule is applied to a few coordinates prevents the use of standard norms to obtain the linear rate, as commonly done for CD methods [43, 45, 46], and (iii) the coordinate sets are overlapping (i.e. non-disjoint), which makes the problem even harder.

Our results also apply to the (primal) parallel distributed setting where multiple workers modify different sets of coordinates of the parameter vector, which is stored in a server accessible by all workers [94–96]. In particular, we show that for this setting the GS selection attains the maximum speedup promised by the theory (see Theorem 13).

Our contributions can be summarized as follows:

- We introduce the class of *set-wise CD* algorithms for asynchronous optimization applicable to both the decentralized and the parallel distributed settings.
- We prove linear convergence rates for strongly convex and smooth f_i and four coordinate (equivalently, neighbor) choices: random uniform, GS, and their variants when the coordinate smoothness constants are known.
- For the cases when these constants are not known, we propose an estimation algorithm based on backtracking [43] for estimating them online, and show that for certain problems this method achieves faster per-iteration convergence than the exact knowledge of these constants.
- To obtain the rates of all considered algorithms, we prove strong convexity in uniquely-defined norms that (i) take into account the graph structure to show strong convexity in the linear subspace where the coordinate updates are applied, and (ii) account for both the random uniform node activation and the application of the GS rule to just a subset of the coordinates.
- We show that the speedup in terms of number of iterations of GS selection with respect to random uniform can be up to $N_{\max}$ (the size of the largest coordinate set).
- We prove that the versions accounting for coordinate smoothness are provably faster than those that do not account for these constants. In particular, we show that the algorithm exploiting both the GS rule *and* the smoothness knowledge is provably faster than all others.
- We support all our results with thorough simulations.

3.2 Related work

A number of algorithms have been proposed to solve (3.1) asynchronously. In [42], the activated node chooses a neighbor uniformly at random and both nodes average their primal local values. In [39] the authors adapted the ADMM algorithm to the decentralized setting, but it was the ADMM of [25] the first one shown to converge at the same rate as the centralized ADMM. The algorithm of [40] tracks the average gradients to converge to the exact optimum instead of just a neighborhood around it, as many algorithms back then. The algorithm of [31] can be used on top of directed graphs, which impose additional challenges. A key novelty of our scheme, compared to this line of work, is that we consider the possibility of letting the nodes *choose smartly* the neighbor to contact in order to make convergence faster.

Work [97] is, to the best of our knowledge, the only work similarly considering smart neighbor selection. The authors propose Max-gossip, a version of the (primal) algorithm in [90] where the activated node averages its local parameter with that of the neighbor with whom the parameter difference is the largest. They show that the algorithm

converges sublineraly to the optimum (for convex functions), and show in numerical simulations that it outperforms random neighbor selection. In contrast, here we propose dual algorithms for which we show linear convergence rates (for smooth and strongly convex functions), and most importantly, (i) we *prove analytically* that either applying the GS rule and/or using the Lipschitz information achieves faster convergence than random neighbor sampling, and (ii) we *quantify* the magnitude of the gains in each case.

Finally, as mentioned earlier, our work relates to standard CD literature. In particular, our theorems extend the results in [45], where the GS rule was shown to be up to d times faster than uniform sampling for $f\colon \mathbb{R}^d \to \mathbb{R}$, to the case where this choice is constrained to a subset of the coordinates only, sets have different sizes, each coordinate belongs to exactly two sets, and sets activate uniformly at random. As we explain in Sections 3.4 and 3.5, these considerations bring important new challenges with respect to the standard single-machine CD algorithms. Furthermore, our algorithms are not only applicable to the decentralized case but also to parallel distributed settings such as [94–96]. For the latter, [98] also analyzed the GS applied to coordinate subsets, but their sets are disjoint, accessible by any worker, and they do not quantify the speedup of the method with respect to random uniform sampling.

3.3 Dual formulation

In this section, we define the notation, obtain the dual problem of (3.2), and analyze the properties of the dual objective function. We will assume throughout that the functions f_i are M_i-smooth and μ_i-strongly convex, i.e. there exist finite constants $M_i \geq \mu_i > 0, i \in [n]$ such that:

$$f_i(y) \leq f_i(x) + \langle \nabla f(x), y - x \rangle + (M_i/2)\|y - x\|_2^2$$
$$f_i(y) \geq f_i(x) + \langle \nabla f(x), y - x \rangle + (\mu_i/2)\|y - x\|_2^2.$$

We define the concatenated primal and dual variables $\theta = [\theta_1^T, \ldots, \theta_n^T]^T \in \mathbb{R}^{nd}$ and $\lambda = [\lambda_1^T, \ldots, \lambda_E^T]^T \in \mathbb{R}^{Ed}$, respectively. The graph's incidence matrix $A \in \mathbb{R}^{n \times E}$ has exactly one 1 and one -1 per column ℓ, in the rows corresponding to nodes $i, j : \ell \equiv (i, j)$, and zeros elsewhere (the choice of sign for each node is irrelevant). We call $u_i \in \mathbb{R}^n$ the vector that has 1 in entry i and 0 elsewhere; we define $e_\ell \in \mathbb{R}^E$ analogously. We use $k \in [K]$ to indicate $k = 1, \ldots, K$. Vectors $\mathbf{1}$ and $\mathbf{0}$ are respectively the all-one and all-zero vectors, and I_d is the $d \times d$ identity matrix. Finally, in order to use matrix operations in the equations below for some operations, we define the block arrays $\Lambda = A \otimes I_d \in \mathbb{R}^{nd \times Ed}$ and $U_i = u_i \otimes I_d \in \mathbb{R}^{nd \times d}$, where $\otimes$ is the Kronecker product. This operation generates arrays analogous to A and u_i where the original entries 1, -1, and 0 have been replaced by I_d, $-I_d$, and the all-zero $d \times d$ matrix, respectively.

We can rewrite now (3.2b) as $\Lambda^T \theta = \mathbf{0}$, and the node variables as $\theta_i = U_i^T \theta$. The minimum

value of (3.2) satisfies:

$$\begin{aligned}\inf_{\theta:\Lambda^T\theta=\mathbf{0}} \sum_{i=1}^{n} f_i(U_i^T\theta) &\overset{\text{(a)}}{=} \inf_{\theta}\sup_{\lambda}\left[\sum_{i=1}^{n} f_i(U_i^T\theta) - \lambda^T\Lambda^T\theta\right] \\ &\overset{\text{(b)}}{=} \sup_{\lambda}\inf_{\theta}\left[\sum_{i=1}^{n} f_i(U_i^T\theta) - \lambda^T\Lambda^T\theta\right] \\ &= -\inf_{\lambda}\sup_{\theta}\sum_{i=1}^{n}\left[(U_i^T\Lambda\lambda)^T U_i^T\theta - f_i(U_i^T\theta)\right] \\ &= -\inf_{\lambda}\sum_{i=1}^{n} f_i^*(U_i^T\Lambda\lambda) \triangleq -\inf_{\lambda} F(\lambda), \end{aligned} \tag{3.3}$$

where (a) holds due to Lagrange duality and (b) holds by strong duality (see e.g. Sec. 5.4 in [80]). Functions f_i^* are the Fenchel conjugates of the f_i, and are defined as

$$f_i^*(y) = \sup_{x\in\mathbb{R}^d}\big(y^T x - f_i(x)\big).$$

Our set-wise CD algorithms converge to the optimal solution of (3.2) by solving (3.3). In particular, they update a single dual variable $\lambda_\ell, \ell \in [E]$ at each iteration and converge to some minimum value λ^* of $F(\lambda)$.

Since $\sum_{i=1}^n f_i(U_i^T\theta)$ in (3.2a) is $M_{\max}$-smooth and $\mu_{\min}$-strongly convex in θ, with $M_{\max} = \max_i M_i$ and $\mu_{\min} = \min_i \mu_i$, function F is L-smooth with $L = \frac{\gamma_{\max}}{\mu_{\min}}$, where $\gamma_{\max}$ is the largest eigenvalue of $\Lambda^+\Lambda$ (Sec. 4 in [38]). We call $\gamma_{\min}^+$ the smallest non-zero eigenvalue[1] of $\Lambda^+\Lambda$.

However, as shown next, function F is *not* strongly convex in the standard L2 norm, which is the property that usually facilitates obtaining linear rates in optimization literature.

Lemma 6. *F is not strongly convex in $\|\cdot\|_2$.*

Proof. Since Λ does not have full column rank in the general case (i.e., unless the graph is a tree), there exist $w \in \mathbb{R}^{Ed}$ such that $w \neq \mathbf{0}$ and $F(\lambda) = F(\lambda + tw)\ \forall t \in \mathbb{R}$. □

Nevertheless, we can still show linear rates for the set-wise CD algorithms using the following result.

Lemma 7 (***Appendix C of*** [99])**.** *F is σ_A-strongly convex in the semi-norm $\|x\|_A \triangleq (x^T\Lambda^+\Lambda x)^{\frac{1}{2}}$, with $\sigma_A = \frac{\gamma_{\min}^+}{M_{\max}}$.*

Above, Λ^+ denotes the pseudo-inverse of Λ. A key fact for the proofs in the next section is that matrix $\Lambda^+\Lambda$ is a projector onto range(Λ^T), the column space of Λ^T.

[1] The "+" stresses that $\gamma_{\min}^+$ is the smallest *strictly positive* eigenvalue.

To simplify the notation, in what follows we assume that $d = 1$, so that $\Lambda = A$, $U_i = u_i$, and the gradient $\nabla_\ell F(\lambda) = \frac{\partial F(\lambda)}{\partial \lambda_\ell}$ of $F(\lambda)$ in the direction of λ_ℓ is a scalar. In Sec. 3.6.3 we discuss how to adapt our proofs to the case $d > 1$.

3.4 Set-wise coordinate descent algorithms

In this section we present the *set-wise CD* algorithms, which can solve generic convex problems (and (3.3) in particular) optimally and asynchronously. In this section, we analyze two possibilities for the coordinate choice within the accessible coordinate subset: (i) sampling uniformly at random (SU-CD), and (ii) applying the GS rule (SGS-CD).

If coordinate ℓ is updated at iteration k and assuming $d = 1$, the standard CD update applied to $F(\lambda)$ is [43]:

$$\lambda^{k+1} = \lambda^k - \eta^k \nabla_\ell F(\lambda^k) e_\ell, \tag{3.4}$$

where η^k is the stepsize. Since $F(\lambda)$ is L-smooth, choosing $\eta^k = 1/L \ \forall k$ guarantees descent at each iteration [45]:

$$F(\lambda^{k+1}) \leq F(\lambda^k) - \frac{1}{2L}\big(\nabla_\ell F(\lambda^k)\big)^2. \tag{3.5}$$

Eq. (3.5) will be the departure point to prove the linear convergence rates of SU-CD and SGS-CD.

We now define formally the set-wise CD algorithms.

Definition 1 (***Set-wise CD algorithm***)**.** In a set-wise CD algorithm, every coordinate $\ell \in [E]$ is assigned to (potentially multiple) sets $\mathcal{S}_i, i \in [n]$, such that all coordinates belong to at least one set. At any time, a set $\mathcal{S}_i$ may activate with uniform probability among the i; the set-wise CD algorithm then chooses a single coordinate $\ell \in \mathcal{S}_i$ to update using (3.4).

The next remark shows how the decentralized problem (3.2) can be solved asynchronously with set-wise CD algorithms.

Remark 1. By letting (i) the E coordinates[2] in Definition 1 be the dual variables λ_ℓ, and (ii) the $\mathcal{S}_i, i \in [n]$ be the sets of dual variables corresponding to the edges that are connected to each node i, nodes can run a set-wise CD algorithm to solve (3.3) (and thus, also (3.2)) asynchronously.

In light of Remark 1, in the following we illustrate the steps that should be performed by the nodes to run the set-wise CD algorithms to find a λ^*. We first note that the gradient

[2] If $d > 1$, the standard CD terminology calls each λ_ℓ a "block coordinate", i.e. a vector of d coordinates out of the $E \cdot d$ coordinates of function $F(\lambda)$.

Table 3.1: Set-related definitions

$\mathcal{S}_i$	Set of edges connected to node i
$\mathcal{N}_i$	Set of neighbors of node i
N_i	Degree of node i, i.e. $N_i = \lvert\mathcal{S}_i\rvert = \lvert\mathcal{N}_i\rvert$
$N_{\max}$	Maximum degree in the network, i.e. $\max_i N_i$
T_i	Selector matrix of set $\mathcal{S}_i$ (see Definition 2)
$\mathcal{S}'_i$	Subset $\mathcal{S}'_i \subseteq \mathcal{S}_i$ such that $\mathcal{S}'_i \cap \mathcal{S}'_j = \emptyset$ if $i \neq j$
T'_i	Selector matrix of set $\mathcal{S}'_i$
$\overline{\mathcal{S}'_i}$	Complement set of $\mathcal{S}'_i$ such that $\overline{\mathcal{S}'_i} = \mathcal{S}_i \setminus \mathcal{S}'_i$
$\overline{T'_i}$	Selector matrix of set $\overline{\mathcal{S}'_i}$

of $F(\lambda)$ in the direction[3] of λ_ℓ for $\ell \equiv (i,j)$ is (computing the gradient of (3.3))

$$\nabla_\ell F(\lambda) = A_{i\ell} \nabla f_i^*(u_i^T A\lambda) + A_{j\ell} \nabla f_j^*(u_j^T A\lambda). \tag{3.6}$$

Nodes can use (3.4) and (3.6) to update the λ_ℓ that they have access to (i.e., those corresponding to the edges they are connected to) as follows: each node i keeps in memory the current values of $\lambda_\ell, \ell \in \mathcal{S}_i$, which are needed to compute $\nabla f_i^*(u_i^T A\lambda)$. Then, when edge $\ell \equiv (i,j)$ is updated (either because node i activated and contacted j, or vice versa), both i and j compute their respective terms in the right-hand side of (3.6) and exchange them through their link. Finally, both nodes compute (3.6) and update their copy of λ_ℓ applying (3.4).

Algorithms 2 and 3 below detail these steps for SU-CD and SGS-CD, respectively. We have used $\mathcal{N}_i$ to indicate the set of neighbors of node i (note that $\mathcal{S}_i = \{\ell : \ell \equiv (i,j), j \in \mathcal{N}_i\}$). Table 3.1 shows this and other set-related notation that will be frequently used in the sections that follow.

We now proceed to describe the SU-CD and SGS-CD algorithms in detail, and prove their linear convergence rates.

3.4.1 Set-wise uniform CD (SU-CD)

In SU-CD, the activated node chooses the neighbor uniformly at random, as shown in Alg. 2. We can compute the per-iteration progress of SU-CD taking expectation in (3.5):

$$\begin{aligned}
\mathbb{E}\Big[F(\lambda^{k+1}) \mid \lambda^k\Big] &\leq F(\lambda^k) - \frac{1}{2L}\mathbb{E}\Big[\big(\nabla_\ell F(\lambda^k)\big)^2 \mid \lambda^k\Big] \\
&= F(\lambda^k) - \frac{1}{2Ln}\sum_{i=1}^{n}\frac{1}{N_i}\sum_{\ell\in\mathcal{S}_i}\big(\nabla_\ell F(\lambda^k)\big)^2 \\
&\leq F(\lambda^k) - \frac{1}{LnN_{\max}}\big\|\nabla F(\lambda^k)\big\|_2^2
\end{aligned} \tag{3.7}$$

[3] This is equivalent to saying "the ℓ-th (block) entry of the gradient $\nabla F(\lambda)$".

Algorithm 2 Set-wise Uniform CD (SU-CD)

Input: Functions f_i, step η^k, incidence matrix A, graph $\mathcal{G}$
Initialize $\theta_i^0, i = 1, \dots, n$ and $\lambda_\ell^0, \ell = 1, \dots, E$
for $k = 1, 2, \dots$ **do**
  Sample activated node $i \in \{1, \dots, n\}$ uniformly
  Node i picks neighbor $j \leftarrow \mathcal{U}\{h : h \in \mathcal{N}_i\}$
  Node i computes $\nabla f_i^*(u_i^T A\lambda)$ and sends it to j
  Node j computes $\nabla f_j^*(u_j^T A\lambda)$ and sends it to i
  Nodes i, j: $(i, j) \equiv \ell$ use (3.6) to update their local copies of λ_ℓ by $\lambda_\ell^k \leftarrow \lambda_\ell^{k-1} - \eta^k \nabla_\ell F(\lambda)$
  $\lambda_m^k \leftarrow \lambda_m^{k-1}$ $\forall$ edges $m \neq \ell$

where $N_i = |\mathcal{S}_i|$, $N_{\max} = \max_i N_i$, and the factor 2 in the denominator disappears because each coordinate $\ell \equiv (i, j)$ is counted twice (once in the sum through $\mathcal{S}_i$ and once in that through $\mathcal{S}_j$).

The standard procedure to show the linear convergence of CD in the single-machine case is to lower-bound $\|\nabla F(\lambda)\|_2^2$ using the strong convexity of the function [43, 45]. However, since F is *not* strongly convex (Lemma 6), we cannot apply this procedure to get the linear rate of SU-CD.

We can, however, use F's strong convexity in $\|\cdot\|_A$ instead (Lemma 7). The next result gives the core of the proof.

Lemma 8. *It holds that*

$$\|\nabla F(\lambda)\|_2 = \|\nabla F(\lambda)\|_A = \|\nabla F(\lambda)\|_A^*, \tag{3.8}$$

where $\|\cdot\|_A^*$ *is the dual norm of* $\|\cdot\|_A$*, defined as (e.g. [80])*

$$\|z\|_A^* = \sup_{x \in \mathbb{R}^d} \left\{ z^T x \,\middle|\, \|x\|_A \leq 1 \right\}. \tag{3.9}$$

Proof. Note that $\forall w \neq \mathbf{0}$ such that $F(\lambda + tw) = F(\lambda)$ $\forall t$, it holds that $w^T \nabla F(\lambda) = 0$ and thus $\nabla F(\lambda) \in \text{range}(A^T)$. This means that $A^+ A \nabla F(\lambda) = I_E \nabla F(\lambda)$, and therefore it holds that $\|\nabla F(\lambda)\|_A = \|\nabla F(\lambda)\|_2$. Finally, since the dual norm of the L2 norm is the L2 norm itself, we have that also $\|\nabla F(\lambda)\|_A^* = \|\nabla F(\lambda)\|_2$, which gives the result. □

We now use Lemma 8 to prove the linear rate of SU-CD.

Theorem 9 (***Rate of SU-CD***)**.** *SU-CD converges as*

$$\mathbb{E}\left[F(\lambda^{k+1}) \mid \lambda^k\right] - F(\lambda^*) \leq \left(1 - \frac{2\sigma_A}{LnN_{\max}}\right)\left[F(\lambda^k) - F(\lambda^*)\right].$$

Proof. Since $F(\lambda)$ is strongly convex in $\|\cdot\|_A$ with strong convexity constant σ_A (Lemma 7), it holds

$$F(y) \geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_A}{2} \|y - x\|_A^2.$$

Algorithm 3 Set-wise Gauss-Southwell CD (SGS-CD)

Input: Functions f_i, step η^k, incidence matrix A, graph $\mathcal{G}$
Initialize $\theta_i^0, i = 1, \dots, n$ and $\lambda_\ell^0, \ell = 1, \dots, E$
for $k = 1, 2, \dots$ **do**
  Sample activated node $i \in \{1, \dots, n\}$ uniformly
  All $h \in \mathcal{N}_i$ compute $\nabla f_h^*(u_h^T A\lambda)$ and send it to i
  Node i computes $\nabla f_i^*(u_i^T A\lambda)$
  Compute $\nabla_\ell F(\lambda)$ $\forall \ell \in \mathcal{S}_i$ (equivalently, $\forall h \in \mathcal{N}_i$) with (3.6) using the received $\nabla f_h^*(u_h^T A\lambda)$
  Node i selects $j \leftarrow \max_{h\in\mathcal{N}_i} |\nabla_\ell F(\lambda)|, \ell \equiv (i,h)$
  Node i sends $\nabla f_i^*(u_i^T A\lambda)$ to j
  Nodes i, j: $(i,j) \equiv \ell$ use (3.6) to update their local copies of λ_ℓ by $\lambda_\ell^k \leftarrow \lambda_\ell^{k-1} - \eta^k \nabla_\ell F(\lambda)$
  $\lambda_m^k \leftarrow \lambda_m^{k-1}$ $\forall$ edges $m \neq \ell$

Minimizing both sides with respect to y as in [45] we get

$$F(x^*) \geq F(x) - \frac{1}{2\sigma_A}(\|\nabla F(x)\|_A^*)^2, \tag{3.10}$$

and rearranging terms we can lower-bound $(\|\nabla F(x)\|_A^*)^2$.

Finally, we can use Lemma 8 to replace $\|\nabla F(x)\|_2^2$ with $(\|\nabla F(x)\|_A^*)^2$ in (3.7), and use the lower bound on $(\|\nabla F(x)\|_A^*)^2$ given by (3.10) to get the result. □

Note that vector λ has $\frac{1}{2}\sum_i N_i = E \leq \frac{nN_{\max}}{2}$ coordinates, where the inequality holds with equality for regular graphs. We make the following remark.

Remark 2. If $\mathcal{G}$ is regular, the linear convergence rate of SU-CD is $\frac{\sigma_A}{LE}$, which matches the rate of single-machine uniform CD for strongly convex functions [43, 45], with the only difference that now the strong convexity constant σ_A is defined over norm $\|\cdot\|_A$.

In the next section we analyze SGS-CD and show that its convergence rate can be up to $N_{\max}$ times that of SU-CD.

3.4.2 Set-wise Gauss-Southwell CD (SGS-CD)

In SGS-CD, as shown in Alg. 3, the activated node i selects the neighbor j to contact applying the GS rule within the edges in $\mathcal{S}_i$:

$$\ell = \operatorname*{argmax}_{m\in\mathcal{S}_i} (\nabla_m F(\lambda))^2,$$

and then j is the neighbor that satisfies $\ell \equiv (i,j)$. In order to make this choice, all nodes $h \in \mathcal{N}_i$ must send their $\nabla f_h^*(u_h^T A\lambda)$ to node i (line 5 in Alg. 3). We discuss this additional communication step of SGS-CD with respect to SU-CD in Sec. 3.8.

To obtain the convergence rate of SGS-CD we will follow the steps taken for SU-CD in the proof of Theorem 9. As done for SU-CD, we start by computing the per-iteration progress taking expectation in (3.5) for SGS-CD:

$$\mathbb{E}\left[F(\lambda^{k+1}) \mid \lambda^k\right] \leq F(\lambda^k) - \frac{1}{2Ln}\sum_{i=1}^{n} \max_{\ell \in \mathcal{S}_i} \left(\nabla_\ell F(\lambda^k)\right)^2. \tag{3.11}$$

Given this per-iteration progress, to proceed as we did for SU-CD we need to show (i) that the sum on the right-hand side of (3.11) defines a norm, and (ii) that strong convexity holds in its dual norm. We start by defining the selector matrices T_i, which will significantly simplify notation.

Definition 2 (***Selector matrices***)**.** The selector matrices $T_i \in \{0,1\}^{N_i \times E}, i = 1, \ldots, n$ select the coordinates of a vector in $\mathbb{R}^E$ that belong to set $\mathcal{S}_i$. Note that any vertical stack of the unitary vectors $\left\{e_\ell^T\right\}_{\ell \in \mathcal{S}_i}$ gives a valid T_i.

We can now show that the sum in (3.11) is a (squared) norm. Since the operation involves applying $\max(\cdot)$ within each set $\mathcal{S}_i$, we will denote this norm $\|x\|_{\text{SM}}$, where the subscript SM stands for "Set-Max".

Lemma 10. *The function* $\|x\|_{SM} \triangleq \sqrt{\sum_{i=1}^n \|T_i x\|_\infty^2} = \sqrt{\sum_{i=1}^n \max_{j \in \mathcal{S}_i} x_j^2}$ *is a norm in* $\mathbb{R}^E$.

Proof. Using $\max_{j \in \mathcal{S}_i}\left(x_j^2 + y_j^2\right) \leq \max_{j \in \mathcal{S}_i} x_j^2 + \max_{j \in \mathcal{S}_i} y_j^2$ and $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ we can show that $\|\cdot\|_{\text{SM}}$ satisfies the triangle inequality. It is straightforward to show that $\|\alpha x\|_{\text{SM}} = |\alpha| \, \|x\|_{\text{SM}}$ and $\|x\|_{\text{SM}} = 0$ if and only if $x = \mathbf{0}$. □

Following the proof of Theorem 9, we would like to show that F is strongly convex in the dual norm $\|\cdot\|_{\text{SM}}^*$. Furthermore, we would like to compare the strong convexity constant σ_{SM} with σ_A to quantify the speedup of SGS-CD with respect to SU-CD. It turns out, though, that computing $\|\cdot\|_{\text{SM}}^*$ is not easy at all; the main difficulty stems from the fact that the sets $\mathcal{S}_i$ are overlapping (or non-disjoint), since each coordinate $\ell \equiv (i,j)$ belongs to both $\mathcal{S}_i$ and $\mathcal{S}_j$. The first scheme in Figure 3.1 illustrates this fact for the 3-node clique.

To circumvent this issue, we define a new norm $\|\cdot\|_{\text{SMNO}}^*$ ("Set-Max Non-Overlapping") that we can directly relate to $\|\cdot\|_{\text{SM}}^*$ (Lemma 11) and whose value we can compute explicitly (Lemma 12), which will later allow us to relate the three strong convexity constants $\sigma_{\text{SM}}, \sigma_{\text{SMNO}}$, and σ_A (Theorem 13).

Definition 3 (***Norm*** $\|\cdot\|_{\textbf{\textit{SMNO}}}^*$)**.** We assume that each coordinate $\ell \equiv (i,j)$ is assigned to only one of the sets $\mathcal{S}_i' \subseteq \mathcal{S}_i$ or $\mathcal{S}_j' \subseteq \mathcal{S}_j$, such that the new sets $\{\mathcal{S}_i'\}_{i=1}^n$ are non-overlapping (some sets can be empty), and all coordinates ℓ belong to exactly one set in

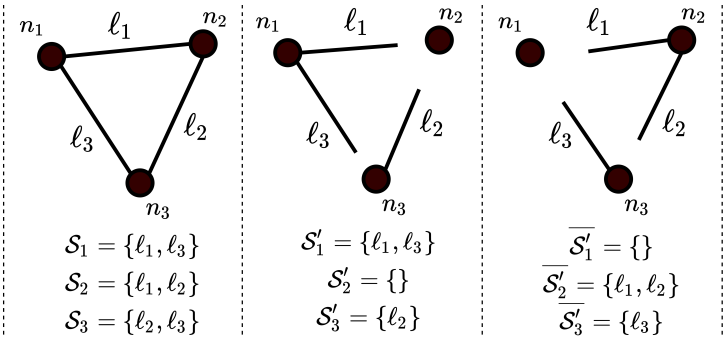

Figure 3.1: Example of sets $\mathcal{S}_i$ and *one possibility* for $\mathcal{S}'_i$ and $\overline{\mathcal{S}'_i}$

$\{\mathcal{S}'_i\}$. We name the selector matrices of these new sets T'_i, so that each possible choice of $\{\mathcal{S}'_i\}$ defines a different set $\{T'_i\}$. Then, we define

$$\|z\|^*_{\text{SMNO}} = \sup_x \left\{ z^T x \;\middle|\; \sqrt{\sum_{i=1}^{n} \|T_i^* x\|_\infty^2} \leq 1 \right\}, \tag{3.12}$$

with the choice of non-overlapping sets

$$\{T_i^*\} = \arg\max_{\{T'_i\}} \sum_{i=1}^{n} \|T'_i x\|_\infty^2. \tag{3.13}$$

Note that the maximizations in (3.12) and (3.13) are coupled. We denote the value of x that attains (3.12) by x^*_{SMNO}.

The definition of sets $\mathcal{S}'_i$ corresponds to assigning each edge ℓ to one of the two nodes at its endpoints, as illustrated in the second scheme of Figure 3.1. Therefore, for each possible pair $(\{\mathcal{S}'_h\}, \{T'_h\}), h \in [n]$ we can define a complementary pair $(\{\overline{\mathcal{S}'_h}\}, \{\overline{T'_h}\})$ such that if $\ell \equiv (i, j)$ was assigned to $\mathcal{S}'_i$ in $\{\mathcal{S}'_h\}$, then it is assigned to $\overline{\mathcal{S}'_j}$ in $\{\overline{\mathcal{S}'_h}\}$. This corresponds to assigning ℓ to the opposite endpoint (node) to the one originally chosen, as shown in the third scheme of Figure 3.1. With these definitions, it holds (potentially with some permutation of the rows):

$$T_i = \begin{bmatrix} T'_i \\ \overline{T'_i} \end{bmatrix} = \begin{bmatrix} T'_i \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \overline{T'_i} \end{bmatrix}, \; i = 1, \ldots, n.$$

We remark that the equality above holds for *any* $\{T'_i\}$ corresponding to a feasible assignment $\{\mathcal{S}'_i\}$, and in particular it hols for $(\{\mathcal{S}^*_i\}, \{T^*_i\})$. This fact is used in the proof of the following lemma, which relates norms $\|\cdot\|^*_{\text{SM}}$ and $\|\cdot\|^*_{\text{SMNO}}$. This will allow us to complete the analysis with $\|\cdot\|^*_{\text{SMNO}}$, which we can compute explicitly (Lemma 12).

Lemma 11. *The dual norm of* $\|\cdot\|_{SM}$*, denoted* $\|\cdot\|^*_{SM}$*, satisfies* $\frac{1}{2}(\|z\|^*_{SMNO})^2 \leq (\|z\|^*_{SM})^2 \leq (\|z\|^*_{SMNO})^2$.

Proof. By definition

$$\|z\|^*_{\text{SM}} = \sup_x \left\{ z^T x \;\middle|\; \sqrt{\sum_{i=1}^{n} \|T_i x\|_\infty^2} \leq 1 \right\}. \tag{3.14}$$

By inspection we can tell that the x that attains the supremum, denoted x^*_{SM}, will satisfy $\sum_{i=1}^n \|T_i x^*_{\text{SM}}\|_\infty^2 = 1$. Similarly, x^*_{SMNO} (defined under (3.13)) must satisfy $\sum_{i=1}^n \|T_i^* x^*_{\text{SMNO}}\|_\infty^2 = 1$. Note that in these two equalities the $\{T_i\}$ are overlapping sets and the $\{T_i^*\}$ are non-overlapping. Therefore, in order to satisfy both equalities it must hold that $|[x^*_{\text{SM}}]_\ell| \leq |[x^*_{\text{SMNO}}]_\ell|, \ell \in [E]$, i.e. the magnitude of the entries of x^*_{SMNO} are equal or larger than the magnitudes of the corresponding entries of x^*_{SM}. Referring to the definitions (3.12) and (3.14), this means that $\|z\|^*_{\text{SM}} \leq \|z\|^*_{\text{SMNO}}$.

We now proceed to show the first inequality in the lemma. We note that

$$\sum_{i=1}^n \|T_i x\|_\infty^2 = \sum_{i=1}^n \left\| \begin{bmatrix} T_i' \\ \mathbf{0} \end{bmatrix} x + \begin{bmatrix} \mathbf{0} \\ \overline{T_i'} \end{bmatrix} x \right\|_\infty^2 \leq \sum_{i=1}^n \|T_i' x\|_\infty^2 + \sum_{i=1}^n \left\|\overline{T_i'} x\right\|_\infty^2 \leq 2 \sum_{i=1}^n \left\|\widehat{T_i'} x\right\|_\infty^2, \quad (3.15)$$

with

$$\{\widehat{T_i'}\} = \arg \max_{\{T_i'\},\{\overline{T_i'}\}} \left(\sum_{i=1}^n \|T_i' x\|_\infty^2, \sum_{i=1}^n \left\|\overline{T_i'} x\right\|_\infty^2 \right). \quad (3.16)$$

We now evaluate (3.15) and (3.16) at x^*_{SMNO}. Due to (3.13) we have $\{\widehat{T_i'}\} = \{T_i^*\}$, and since $\sum_{i=1}^n \|T_i^* x^*_{\text{SMNO}}\|_\infty^2 = 1$, the rightmost member of (3.15) takes value 2. Then, dividing both sides of (3.15) by 2 we obtain

$$\frac{1}{2} \sum_{i=1}^n \|T_i x^*_{\text{SMNO}}\|_\infty^2 = \sum_{i=1}^n \left\| T_i \frac{x^*_{\text{SMNO}}}{\sqrt{2}} \right\|_\infty^2 \leq 1,$$

and since $\sum_{i=1}^n \|T_i x^*_{\text{SM}}\|_\infty^2 = 1$, we conclude that it must hold that $\frac{1}{\sqrt{2}} |[x^*_{\text{SMNO}}]_\ell| \leq |[x^*_{\text{SM}}]_\ell|, \ell \in [E]$, and thus $\frac{1}{\sqrt{2}} \|z\|^*_{\text{SMNO}} \leq \|z\|^*_{\text{SM}}$. □

The next lemma gives the value of $\|x\|^*_{\text{SMNO}}$ explicitly, which will be needed to compare the strong convexity constant σ_{SMNO} with σ_A.

Lemma 12. *It holds that* $\|x\|^*_{SMNO} = \sqrt{\sum_{i=1}^n \|T_i^* x\|_1^2}$.

Proof. Since the sets $\{\mathcal{S}_i^*\}$ are non-overlapping and in (3.12) norm $\|\cdot\|_\infty$ is applied per-set, the entries x_ℓ of x^*_{SMNO} will have $|x_\ell| = x^{(i)} \geq 0 \; \forall \; \ell \in \mathcal{S}_i^*$ and the sign will match that of the entries of z, i.e. $\text{sign}(x_\ell) = \text{sign}(z_\ell)$. The maximization of (3.12) then becomes

$$\begin{aligned} \underset{\{x^{(i)}\}}{\text{maximize}} \quad & \sum_{i=1}^n \sum_{\ell \in \mathcal{S}_i^*} \left(|z_\ell| \cdot x^{(i)} \right) \\ \text{s.t.} \quad & \sqrt{\sum_{i=1}^n \left(x^{(i)}\right)^2} \leq 1. \end{aligned}$$

Factoring out $x^{(i)}$ in the objective and noting that $\sum_{\ell \in \mathcal{S}_i^*} |z_\ell| = \|T_i^* z\|_1$, we can define $w = [x^{(1)}, \ldots, x^{(n)}]^T$ and $y = [\|T_1^* z\|_1, \ldots, \|T_n^* z\|_1]^T$ so that (3.12) now reads

$$\|z\|_{\text{SMNO}}^* = \sup_w \left\{ y^T w \;\middle|\; \|w\|_2 \leq 1 \right\}.$$

The right-hand side is the definition of $\|\cdot\|_2^*$, the dual of the L2 norm, evaluated at y. Since $\|\cdot\|_2^* = \|\cdot\|_2$, we have that $\|z\|_{\text{SMNO}}^* = \|y\|_2 = \sqrt{\sum_{i=1}^n \|T_i^* z\|_1^2}$. □

We can now prove the linear convergence rate of SGS-CD.

Theorem 13 (***Rate of SGS-CD***). *SGS-CD converges as*

$$E_{F(\lambda^{k+1})|\lambda^k}[-]F(\lambda^*) \leq \left(1 - \frac{\sigma_{SM}}{Ln}\right)\left[F(\lambda^k) - F(\lambda^*)\right],$$

with

$$\frac{\sigma_A}{N_{\max}} \leq \sigma_{SM} \leq 2\sigma_A. \tag{3.17}$$

Proof. Similarly to what we did for SU-CD, we can depart from the strong convexity of F in the $\|\cdot\|_{\text{SM}}$ norm:

$$F(y) \geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_{\text{SM}}}{2}(\|y - x\|_{\text{SM}}^*)^2,$$

then minimize both sides with respect to y to obtain

$$F(x^*) \geq F(x) - \frac{1}{2\sigma_{\text{SM}}}(\|\nabla F(x)\|_{\text{SM}})^2, \tag{3.18}$$

which is analogous to (3.10), and then rearrange terms to obtain a lower bound on $\|\nabla F(\lambda)\|_{\text{SM}}^2$. Using this lower bound in (3.11) gives the rate of SGS-CD.

Since this rate is given in terms of σ_{SM} and that of SU-CD in Theorem 9 is given in terms of σ_A, we need (3.17) to compare both rates. However, we cannot prove these inequalities directly because we cannot compare norms $\|\cdot\|_A$ and $\|\cdot\|_{\text{SM}}^*$ (due to the overlap of the coordinate sets, which prevents us from computing the latter). However, we *can* compare $\|\cdot\|_A$ with $\|\cdot\|_{\text{SMNO}}^*$ and $\|\cdot\|_{\text{SM}}^*$ with $\|\cdot\|_{\text{SMNO}}^*$ individually, from which we will obtain (3.17). In particular, we will show the inequalities

$$\frac{\sigma_A}{N_{\max}} \leq \sigma_{\text{SMNO}} \leq \sigma_A \tag{3.19}$$

and

$$\sigma_{\text{SMNO}} \leq \sigma_{\text{SM}} \leq 2\sigma_{\text{SMNO}}. \tag{3.20}$$

We start by proving (3.19). Below we assume $x \in \text{range}(A^T)$; the results here can then be directly applied to the proofs above because $\|\cdot\|_A, \|\cdot\|_{\text{SM}}, \|\cdot\|_{\text{SMNO}}$ and their duals are applied to ∇F, which is always in $\text{range}(A^T)$ (Lemma 8).

For $x \in \text{range}(A^T)$ it holds that (Lemmas 8 and 12):

$$\|x\|_A^2 = \|x\|_2^{\,2} = \sum_{i=1}^{E} x_i^2 = \sum_{i=1}^{n} \|T_i^* x\|_2^{\,2}$$
$$\left(\|x\|_{\text{SMNO}}^*\right)^2 = \sum_{i=1}^{n} \|T_i^* x\|_1^2.$$

We also note that, using the Cauchy-Schwarz inequality and denoting $[v]_i$ the i^{th} entry of vector v, it holds both that

$$\sum_{i=1}^{n} \|T_i^* x\|_2^{\,2} \leq \sum_{i=1}^{n} \left(\sum_{j \in \mathcal{S}_i^*} |x_j| \right)^2 = \sum_{i=1}^{n} \|T_i^* x\|_1^2, \text{ and}$$
$$\sum_{i=1}^{n} \|T_i^* x\|_1^2 = \sum_{i=1}^{n} \left(\mathbf{1}^T \left[\left| [T_i^* x]_1 \right|, \ldots, \left| [T_i^* x]_{N_i^*} \right| \right]^T \right)^2$$
$$\overset{\text{C.S.}}{\leq} \sum_{i=1}^{n} N_i^* \|T_i^* x\|_2^{\,2} \leq N_{\max} \sum_{i=1}^{n} \|T_i^* x\|_2^{\,2},$$

where $N_i^* = |\mathcal{S}_i^*|$. We can summarize these relations as

$$\frac{1}{N_{\max}} \left(\|x\|_{\text{SMNO}}^*\right)^2 \leq \|x\|_A^2 \leq \left(\|x\|_{\text{SMNO}}^*\right)^2.$$

Using these inequalities in the strong convexity definitions, similarly to [45], we get both

$$\begin{aligned} F(y) &\geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_A}{2} \left(\|y - x\|_A\right)^2 \\ &\geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_A}{2N_{\max}} \left(\|y - x\|_{\text{SMNO}}^*\right)^2, \end{aligned} \tag{3.21}$$

and

$$\begin{aligned} F(y) &\geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_{\text{SMNO}}}{2} \left(\|y - x\|_{\text{SMNO}}^*\right)^2 \\ &\geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_{\text{SMNO}}}{2} \left(\|y - x\|_A\right)^2. \end{aligned} \tag{3.22}$$

Equation (3.21) says that F is at least $\frac{\sigma_A}{N_{\max}}$-strongly convex in $\|\cdot\|_{\text{SMNO}}^*$, and eq. (3.22) says that F is at least σ_{SMNO}-strongly convex in $\|\cdot\|_A$. Together they imply (3.19).

We can show (3.20) by the same procedure applied in eqs. (3.21) and (3.22), but now using the strong convexity of F in norms $\|\cdot\|_{\text{SM}}^*$ and $\|\cdot\|_{\text{SMNO}}^*$ together with Lemma 11. From $\frac{1}{2}\left(\|z\|_{\text{SMNO}}^*\right)^2 \leq \left(\|z\|_{\text{SM}}^*\right)^2$ we get $\frac{1}{2}\sigma_{\text{SM}} \leq \sigma_{\text{SMNO}}$, and from $\left(\|z\|_{\text{SM}}^*\right)^2 \leq \left(\|z\|_{\text{SMNO}}^*\right)^2$ we get $\sigma_{\text{SMNO}} \leq \sigma_{\text{SM}}$.

Finally, putting (3.19) and (3.20) together gives (3.17). □

Theorems 9 and 13 together allow us to compare the convergence rates of SU-CD and SGS-CD. We note that when σ_{SM} takes the upper value in (3.17), SGS-CD is (in expectation) $N_{\max}$ times faster than SU-CD. The lower bound in (3.17), on the other hand,

suggests that SGS-CD could be slower than SU-CD. We remark that (in expectation) this is not true and the lower bound is vacuous, since the following holds.

Remark 3. For the same sequence of node activations, the suboptimality reduction of SGS-CD at each iteration is equal to or larger than that of SU-CD.

Taking this fact into account, we have the following result.

Corollary 3. *In expectation, SGS-CD converges at least as fast as SU-CD, and can be up to $N_{\max}$ times faster.*

Note that this result is analogous to that of [45] for single-machine CD, where they show that the GS rule can be up to d times faster than uniform sampling, d being the dimensionality of the problem.

We remark that achieving the upper bound of $N_{\max}$ speedup may require designing a scenario particularly favorable to SGS-CD with respect to SU-CD. Similarly, finding a setting where the former converges at the same speed as the latter also requires designing a particularly adversarial setting.

In our simulations of Section 3.7 for the decentralized setting, SGS-CD achieves a speedup approximately in the middle of the range between 1 and $N_{\max}$. We show that this speedup *increases linearly with* $N_{\max}$, achieving remarkable gains in terms of suboptimality reduction versus number of iterations (see Fig. 3.2). Furthermore, in the same figure we show that for the parallel distributed setting *the maximum speedup of* $N_{\max}$ *is attainable*. We explain this further in Section 3.6.1.

3.5 Set-wise Lipschitz CD algorithms

In Section 3.4 we stated that the dual function F is L-smooth and therefore a sufficient condition for the set-wise algorithms to converge was using stepsize $\eta^k = 1/L$. However, the updates of some (and maybe *many*) coordinates could use larger stepsizes by exploiting the fact F has coordinate-wise smoothness $L_\ell \leq L$, i.e. for $\alpha \in \mathbb{R}$:

$$|\nabla_\ell F(\lambda + \alpha e_\ell) - \nabla_\ell F(\lambda)| \leq L_\ell \alpha. \tag{3.23}$$

Therefore, when the coordinate-wise Lipschitz constants L_ℓ are known or can be estimated (see Section 3.5.3) we can apply the update (3.4) with stepsize $\eta^k = 1/L_\ell$, with ℓ being the coordinate updated at iteration k.

In the sections that follow we show that by using the knowledge (or estimation) of the coordinate-wise Lipschitz constants and per-coordinate stepsizes we can have:

1. An algorithm that has randomized but non-uniform neighbor selection that is provably faster than SU-CD. We call this algorithm Set-wise Lipschitz CD (SL-CD).

2. An algorithm that applies locally the Gauss-Southwell Lipschitz rule [45] and that converges provably faster than both SL-CD and SGS-CD. We call this algorithm Set-wise GS Lipschitz CD (SGSL-CD).

Once again, while the seminal work of [45] has analyzed both of these rules in the context of single-machine coordinate descent, their adaptation to set-wise CD brings important new challenges. In this section, we prove that SL-CD is at least as fast as SU-CD, and that SGSL-CD is at least as fast as the fastest algorithm between SGS-CD and SL-CD.

In the proofs that follow we will use the following fact.

Fact 1. Denote $a \circ b$ the per-entry product of vectors a and b. Then, for any norm $\|\cdot\|$ and finite $a : a_i > 0 \ \forall i$, if we define $\|x\|_a := \|a \circ x\|$, then $\|x\|_a^* := \|a_{-1} \circ x\|^*$ with $a_{-1} = [\frac{1}{a_1}, \dots, \frac{1}{a_d}]$.

Proof. By definition

$$\|z\|_a^* = \sup_{\|x\|_a \leq 1} z^T x,$$

and defining $y := a \circ x$ we get

$$\|z\|_a^* = \sup_{\|y\| \leq 1} z^T (a_{-1} \circ y) = \|a_{-1} \circ z\|^*.$$

□

3.5.1 Set-wise Lipschitz CD (SL-CD)

In SL-CD, an activated node i chooses the edge $\ell \in \mathcal{S}_i$ to update at random with probability

$$p_\ell = \frac{L_\ell}{\sum_{m \in \mathcal{S}_i} L_m} \tag{3.24}$$

and updates λ_ℓ applying (3.4) with stepsize $\eta^k = 1/L_\ell$, where ℓ is the chosen edge.

For convenience, we define the quantities

$$L^{(i)} := \sum_{m \in \mathcal{S}_i} L_m$$

and

$$\mathcal{L}_\ell := \left(\frac{1}{L^{(i)}} + \frac{1}{L^{(j)}} \right) \quad \text{for} \quad \ell \equiv (i, j).$$

With these definitions, and taking expectation in (3.5) for the Lipschitz-dependent neigh-

bor sampling probabilities (3.24) gives

$$\begin{aligned}\mathbb{E}\big[F(\lambda^{k+1}) \mid \lambda^k\big] &\leq F(\lambda^k) - \frac{1}{2}\mathbb{E}\left[\frac{1}{L_{\ell_k}}\big[\nabla_{\ell_k} F(\lambda^k)\big]^2\right] \\ &= F(\lambda^k) - \frac{1}{2n}\sum_{i=1}^{n}\frac{1}{L^{(i)}}\sum_{\ell\in\mathcal{S}_i}\big[\nabla_\ell F(\lambda^k)\big]^2 \\ &\overset{(a)}{=} F(\lambda^k) - \frac{1}{2n}\sum_{\ell=1}^{E}\mathcal{L}_\ell\big[\nabla_\ell F(\lambda^k)\big]^2\end{aligned}$$

where in (a) we used that $\ell \equiv (i,j)$ implies $\ell \in \mathcal{S}_i, \mathcal{S}_j$.

In order to prove the convergence rate of SL-CD, provided in Theorem 14, we define the norm

$$\|x\|_{\mathcal{L}} := \sqrt{\sum_{\ell=1}^{E}\mathcal{L}_\ell x_\ell^2},$$

so that we can write the per-iteration progress of SL-CD as

$$\mathbb{E}[F(\lambda^{k+1})] \leq F(\lambda^k) - \frac{1}{2n}\big\|\nabla F(\lambda^k)\big\|_{\mathcal{L}}^2. \tag{3.25}$$

Noting that $\|x\|_{\mathcal{L}} = \big\|x \circ \big[\sqrt{\mathcal{L}_1}, \ldots, \sqrt{\mathcal{L}_E}\big]\big\|_2$ we can apply Fact 1 to get its dual norm:

$$\|x\|_{\mathcal{L}}^* = \sqrt{\sum_{\ell=1}^{E}\frac{1}{\mathcal{L}_\ell}x_\ell^2}.$$

We call $\sigma_{\mathcal{L}}$ the strong convexity constant of F in this norm:

$$F(y) \geq F(x) + \langle \nabla F(x), y - x\rangle + \frac{\sigma_{\mathcal{L}}}{2}(\|y-x\|_{\mathcal{L}}^*)^2. \tag{3.26}$$

We use the definitions of $\|\cdot\|_{\mathcal{L}}^*$ and $\sigma_{\mathcal{L}}$ in the proof of the linear rate of SL-CD, given in the theorem below.

Theorem 14 (***Rate of SL-CD***). *SL-CD converges as*

$$\mathbb{E}\big[F(\lambda^{k+1}) \mid \lambda^k\big] - F(\lambda^*) \leq \left(1 - \frac{\sigma_{\mathcal{L}}}{n}\right)\big[F(\lambda^k) - F(\lambda^*)\big]$$

and it holds that

$$\sigma_A\mathcal{L}_{\min} \leq \sigma_{\mathcal{L}} \leq \sigma_A\mathcal{L}_{\max} \tag{3.27}$$

with $\mathcal{L}_{\min} = \min_\ell \mathcal{L}_\ell$ *and* $\mathcal{L}_{\max} = \max_\ell \mathcal{L}_\ell$.

Proof. We start by proving the linear rate. Minimizing both sides of (3.26) with respect to y as done in (3.10) and (3.18) we get

$$F(x^*) \geq F(x) - \frac{1}{2\sigma_{\mathcal{L}}}\|\nabla F(x)\|_{\mathcal{L}}^2.$$

Rearranging terms gives a lower bound on $\|\nabla F(x)\|_{\mathcal{L}}^2$, and replacing in (3.25) gives the result.

We now move on to show (3.27). Once again, since the norms are evaluated at $\nabla F(\lambda)$ and (3.8) holds, to obtain the relation between $\sigma_{\mathcal{L}}$ and σ_A we will compare $\|\cdot\|_{\mathcal{L}}^*$ with $\|\cdot\|_2$ directly. We have that

$$c\|x\|_2^2 - (\|x\|_{\mathcal{L}}^*)^2 = c\sum_\ell x_\ell^2 - \sum_\ell \frac{1}{\mathcal{L}_\ell}x_\ell^2 = \sum_\ell \left(c - \frac{1}{\mathcal{L}_\ell}\right)x_\ell^2.$$

For $c \geq \max_\ell \frac{1}{\mathcal{L}_\ell} = \frac{1}{\mathcal{L}_{\min}}$ the expression above is larger than zero, and thus

$$\frac{1}{\mathcal{L}_{\min}}\|x\|_2^2 \geq (\|x\|_{\mathcal{L}}^*)^2. \tag{3.28}$$

Similarly, we have that

$$c\|x\|_{\mathcal{L}}^2 - \|x\|_2^2 = \sum_\ell \left(\frac{c}{\mathcal{L}_\ell} - 1\right)x_\ell^2$$

is larger than zero for $c \geq \mathcal{L}_{\max}$, and therefore

$$\mathcal{L}_{\max}(\|x\|_{\mathcal{L}}^*)^2 \geq \|x\|_2^2. \tag{3.29}$$

Using these inequalities (and Lemma 8) in the strong convexity definitions we have on the one hand:

$$\begin{aligned} f(y) &\geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\sigma_A}{2}\|y - x\|_A^2 \\ &\geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\sigma_A \mathcal{L}_{\min}}{2}(\|y - x\|_{\mathcal{L}}^*)^2, \end{aligned} \tag{3.30}$$

and on the other hand:

$$\begin{aligned} f(y) &\geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\sigma_{\mathcal{L}}}{2}(\|y - x\|_{\mathcal{L}}^*)^2 \\ &\geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\sigma_{\mathcal{L}}}{2\mathcal{L}_{\max}}\|y - x\|_A^2. \end{aligned} \tag{3.31}$$

Eqs. (3.30) and (3.31) indicate respectively that $\sigma_{\mathcal{L}} \geq \sigma_A \mathcal{L}_{\min}$ and that $\sigma_A \geq \frac{\sigma_{\mathcal{L}}}{\mathcal{L}_{\max}}$. Putting both together gives (3.27). □

Having obtained the rate of SL-CD, we can compare it against that of SU-CD. We have the following result.

Corollary 4. *SL-CD converges as fast or faster than SU-CD.*

Proof. The convergence rate of SU-CD is $\frac{2\sigma_A}{LnN_{\max}}$ (Theorem 9) and that of SL-CD is $\frac{\sigma_{\mathcal{L}}}{n}$ (Theorem 14). Since in the slowest case of SL-CD we have $\sigma_{\mathcal{L}} = \sigma_A \mathcal{L}_{\min}$, it suffices to show that $\mathcal{L}_{\min} \geq \frac{2}{LN_{\max}}$. Indeed, we have that

$$L^{(i)} = \sum_{\ell \in \mathcal{S}_i} L_\ell \leq L_{\max}|\mathcal{S}_i| \leq L_{\max}N_{\max}$$

and therefore

$$\mathcal{L}_{\min} = \min_{(i,j)\in\mathcal{E}} \left(\frac{1}{L^{(i)}} + \frac{1}{L^{(j)}} \right) \geq \frac{2}{L_{\max} N_{\max}}$$

The proof is complete by noting that it always holds that $L_{\max} \leq L$ [44]. □

Since both SL-CD and SGS-CD can converge at the same speed as SU-CD in the worst case, we cannot claim that either of them is faster than the other. We can, however, exploit the knowledge of the Lipschitz constants to get an improved version of the GS rule, known as the Gauss-Southwell Lipschitz rule [45], that when combined with per-coordinate stepsizes allows for faster convergence than both SGS-CD *and* SL-CD. We call this algorithm SGSL-CD, and we analyze it next.

3.5.2 Set-wise Gauss-Southwell Lipschitz CD (SGSL-CD)

If node i goes active, the Gauss-Southwell Lipschitz (GSL) rule chooses to update $\lambda_\ell, \ell \in \mathcal{S}_i$ according to

$$\ell = \operatorname*{argmax}_{m\in\mathcal{S}_i} \frac{\left|\nabla_m f(x^k)\right|}{\sqrt{L_m}}.$$

If we now use the GSL rule with the per-coordinate stepsizes $\eta^k = 1/L_\ell$, the per-iteration progress given by (3.5) becomes:

$$\mathbb{E}F(\lambda^{k+1}) \leq F(\lambda^k) - \frac{1}{2n} \sum_{i=1}^{n} \max_{\ell\in\mathcal{S}_i} \left(\frac{1}{L_\ell} \left[\nabla_\ell F(\lambda^k)\right]^2 \right). \tag{3.32}$$

Note the resemblance of this expression with the per-iteration progress of SGS-CD in (3.11). Similarly to the previous procedures, we define the "Set-Max Lipschitz" norm:

$$\|x\|_{\text{SML}} := \sqrt{\sum_{i=1}^{n} \max_{\ell\in\mathcal{S}_i} \left(\frac{1}{L_\ell} x_\ell^2 \right)}, \tag{3.33}$$

and call σ_{SML} the strong convexity constant of F in the dual norm $\|\cdot\|_{\text{SML}}^*$

$$F(y) \geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_{\text{SML}}}{2} (\|y - x\|_{\text{SML}}^*)^2. \tag{3.34}$$

We can now state the convergence rate of SGSL-CD.

Theorem 15 (***Rate of SGSL-CD***). *SGSL-CD converges as*

$$E_{F(\lambda^{k+1})|\lambda^k}[-]F(\lambda^*) \leq \left(1 - \frac{\sigma_{SML}}{n}\right) \left[F(\lambda^k) - F(\lambda^*)\right], \tag{3.35}$$

and it holds that

$$\frac{\sigma_{SM}}{L} \leq \sigma_{SML}. \tag{3.36}$$

Proof. The expression of the rate is obtained with the procedure followed for the previous algorithms: minimizing both sides of (3.34) with respect to y and arranging terms we can find $\|\nabla F(x)\|_{\text{SML}} \geq 2\sigma_{\text{SML}}(F(x) - F(x^*))$, and using this in (3.32) gives (3.35).

We now show (3.36). By definition, the dual norm of $\|\cdot\|_{\text{SML}}$ is

$$\|z\|^*_{\text{SML}} := \sup_x \left\{ z^T x \;\middle|\; \|x\|_{\text{SML}} \leq 1 \right\} = \sup_x \left\{ z^T x \;\middle|\; \sqrt{\sum_{i=1}^n \max_{\ell \in \mathcal{S}_i} \left(\frac{1}{L_\ell} x_\ell^2 \right)} \leq 1 \right\}.$$

Similarly, we can write the dual norm of $\|\cdot\|_{\text{SM}}$ provided in (3.14) also as

$$\|z\|^*_{\text{SM}} := \sup_x \left\{ z^T x \;\middle|\; \|x\|_{\text{SM}} \leq 1 \right\} = \sup_x \left\{ z^T x \;\middle|\; \sqrt{\sum_{i=1}^n \max_{\ell \in \mathcal{S}_i} x_\ell^2} \leq 1 \right\}.$$

We call the values that achieve the supremum x^*_{SML} and x^*_{SM}, respectively. To maximize $z^T x$, these values will satisfy the constraints of each dual norm with equality, i.e.

$$\|x^*_{\text{SML}}\|_{\text{SML}} = 1 \quad \text{and} \quad \|x^*_{\text{SM}}\|_{\text{SM}} = 1.$$

From these conditions and the definitions of the dual norms above we obtain

$$x^*_{\text{SML}} \circ \left[\frac{1}{\sqrt{L_1}}, \cdots, \frac{1}{\sqrt{L_E}} \right] = x^*_{\text{SM}}.$$

Furthermore, using again $L_{\max} \leq L$, we have

$$x^*_{\text{SM}} = x^*_{\text{SML}} \circ \left[\frac{1}{\sqrt{L_1}}, \cdots, \frac{1}{\sqrt{L_E}} \right] \succeq \frac{1}{\sqrt{L_{\max}}} x^*_{\text{SML}} \succeq \frac{1}{\sqrt{L}} x^*_{\text{SML}},$$

where "$\succeq$" indicates coordinate-wise inequality, and therefore

$$\|z\|^*_{\text{SM}} \geq \frac{1}{\sqrt{L}} \|z\|^*_{\text{SML}}.$$

Lastly, using this inequality in the strong convexity equation of F in $\|\cdot\|^*_{\text{SM}}$:

$$\begin{aligned} F(y) &\geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_{\text{SM}}}{2} (\|y - x\|^*_{\text{SM}})^2 \\ &\geq F(x) + \langle \nabla F(x), y - x \rangle + \frac{\sigma_{\text{SM}}}{2L} (\|y - x\|^*_{\text{SML}})^2, \end{aligned}$$

from where we obtain $\frac{\sigma_{\text{SM}}}{L} \leq \sigma_{\text{SML}}$ □

Theorem 15 states that SGSL-CD converges (in expectation) at least as fast as SGS-CD. Algorithm SGSL-CD is also at least as fast as SL-CD by an argument analogous to Remark 3: for the same sequence of node activations, the set-wise GSL rule achieves an equal or larger suboptimality reduction than the random coordinate sampling with the probabilities in (3.24). We thus have the following result.

Corollary 5. *In expectation, SGSL-CD converges equally fast or faster than both SGS-CD and SL-CD.*

We remark that we could have compared the convergence rates of SGSL-CD and SL-CD following a procedure similar to the one used to compare SGS-CD and SU-CD, where we would define a norm using non-overlapping sets (in this case, accounting also for the coordinate-wise Lipschitz consants) as an intermediate step to compare the strong convexity constants $\sigma_{\mathcal{L}}$ and σ_{SML}. We did this derivation and observed that just as it happened with σ_{SM} in eq. (3.17), the lower bound on σ_{SML} is not tight and suggests that SL-CD could be faster than SGSL-CD, which as we argued above, is not true.

Corollary 4 states that SL-CD is faster than SU-CD, and Corollary 5 states that SGSL-CD is the fastest of all algorithms analyzed here. However, these two methods depend on the knowledge of the per-coordinate Lipschitz constants L_ℓ (see eq. (3.23)). These constants are the global upper bounds on the diagonal entries of the Hessian $H = \nabla^2 F$, given by

$$H_{\ell\ell}(\lambda) = \nabla^2 f_i^*(U_i^T \Lambda\lambda) + \nabla^2 f_j^*(U_j^T \Lambda\lambda),\ \ell \equiv (i,j),$$

i.e. $H_{\ell\ell}(\lambda) \leq L_\ell\ \forall\lambda$. We next describe a decentralized algorithm to estimate these values when they are not known. In Section 3.7 we show that the versions of SL-CD and SGSL-CD that use estimated constants, which we call SeL-CD and SGSeL-CD, still perform remarkably well.

3.5.3 Smoothness constants estimation

In [43] the author proposed a method to estimate the value of the instantaneous Lipschitz constants $L_\ell(\lambda)$ when they are not known. By *instantaneous* we mean the value of the Lipschitz constants at the current point λ^k, and not global values valid for any value of λ.

The procedure consists on finding, every time that variable λ_ℓ is going to be updated at iteration k, the *lowest* value $L_\ell(\lambda^k)$ such that after applying update (3.4) with stepsize $\eta^k = 1/L_\ell(\lambda^k)$ it holds that $\nabla_\ell F(\lambda^k) \cdot \nabla_\ell F(\lambda^{k+1}) > 0$. In other words, the procedure searches for a Lipschitz constant (or equivalently, a stepsize) for which the update (3.4) does not *overshoot*, making the gradient take a completely different direction.

The procedure to estimate $L_\ell(\lambda^k)$ is shown in Algorithm 4. In our numerical simulations, we denote SeL-CD and SGSeL-CD the versions of SL-CD and SGSL-CD that use estimated Lipschitz constants instead of the exact L values. SeL-CD is obtained by replacing line 8 in Alg. 2 with Alg. 4 and using the estimated values L_ℓ for the random sampling. SGSeL-CD is obtained by replacing line 10 in Alg. 3 with Alg. 4 and using the estimated values L_ℓ for the GSL neighbor choice.

The choice of the initial value $\widehat{L}_\ell^0$ before entering the search loop is subject to a trade-off: if $\widehat{L}_\ell^0$ is too big, the loop will be exited after only one iteration but we risk being too conservative and making a much smaller step than we could. Conversely, if $\widehat{L}_\ell^0$ is too small, by repeated doubling we will eventually find the value $\widehat{L}_\ell$ that is closest to the

Algorithm 4 Online smoothness constant estimation

Assumption: Nodes i and j will update $\lambda_\ell, \ell \equiv (i,j)$ and they have already exchanged their $\nabla f_x^*(u_x^T A\lambda), x = i, j$.
Inputs: Instantaneous smoothness starting value $\widehat{L}_\ell^0$
Each node $x = i, j$ **then runs:**
Compute $\nabla_\ell F(\lambda^k)$ with (3.6) using $\nabla f_x^*(u_x^T A\lambda), x = i, j$
Set $\widehat{L}_\ell \leftarrow \widehat{L}_\ell^0$
do ...
  Set $\widehat{L}_\ell \leftarrow 2 \cdot \widehat{L}_\ell$
  Compute $\widehat{\lambda}_\ell = \lambda_\ell^k - (1/\widehat{L}_\ell) \cdot \nabla_\ell F(\lambda^k)$
  Compute $\nabla f_x^*(u_x^T A\widehat{\lambda}_\ell)$ and send to neighbor
  Compute $\nabla_\ell F(\widehat{\lambda})$ with (3.6) using $\nabla f_x^*(u_x^T A\widehat{\lambda}_\ell), x{=}i, j$
  ... while $\nabla_\ell F(\lambda^k) \cdot \nabla_\ell F(\widehat{\lambda}) \leq 0$
end do-while
Set $L_\ell^{k+1} \leftarrow \widehat{L}_\ell$ and $\lambda_\ell^{k+1} \leftarrow \widehat{\lambda}_\ell$

true instantaneous smoothness $L_\ell(\lambda^k)$, but this may take many iterations inside the loop, which means many rounds of computation and communication for the nodes involved.

How are these estimated values expected to perform with respect to the analytical ones? This depends very much on the problem at hand. We can easily construct a case (for where the exact constants perform better than the estimated: assume that the function to optimize is $f(x) = x^T \mathrm{diag}(L_1, \ldots, L_d)x$. If $x^0 \neq \mathbf{0}$, then the algorithm using the analytic constants converges in d steps (one in each coordinate). However, using estimated constants will most likely exit the search loop finding values $\widehat{L}_i \neq L_i$, and thus will need more iterations. Conversely, the analytical constants are *global* quantities, and therefore, although they are valid in the complete optimization space, they might be very different to the real instantaneous Lipschitz constants for many values of λ. In that case, we may get a much better approximation to the instantaneous value using the estimations, and therefore a faster convergence due to using a larger stepsize. In Section 3.7 we provide numerical tests where we observe both behaviors.

3.6 Additional considerations

3.6.1 Application to parallel distributed optimization

In the parallel distributed setup, the parameter vector is stored in a server accessible by multiple workers, each of which modifies some or all of the coordinates of the parameter. We assume that coordinates are updated by a single worker at each iteration and workers always access the most recent value of the parameter.

In this setting, if there are E coordinates, n workers, and we let each worker i update a different set $\mathcal{S}_i$ of coordinates such that (i) the sets overlap, and (ii) each coordinate can

be updated by exactly two workers, then all results presented previously (Theorems 9, 13, 14, and 15) hold also for this setting. We remark these two conditions are *not* necessary conditions to apply the set-wise CD algorithms to the parallel distributed setting, but only to directly apply the results of the theorems, which were derived for the decentralized setting. In fact, the family of set-wise CD algorithms *can always be applied* to the parallel distributed setting *independently* of the degree of overlapping of the sets and the number of the coordinates modified by each worker.

We can then also easily construct a setting where SGS-CD is $N_{\max}$ times faster than SU-CD: let all sets have the same size $|\mathcal{S}_i| = N_{\max}\ \forall i$, exactly $(N_{\max} - 1)$ coordinates in each set have $\nabla_m F(\lambda) = 0$, and only one ℓ have $\nabla_\ell F(\lambda) \neq 0$. In this case, on average *only* $\frac{1}{N_{\max}}$ *times will SU-CD choose the coordinate that gives some improvement*, while SGS-CD will do it at all iterations.

Note that achieving the maximum speedup for this carefully crafted scenario requires that the gradients of all coordinates are independent, which is not verified in the decentralized optimization setting: according to eq. (3.6), for a $\nabla_m F$ to be zero, it must hold that $\nabla f_i^* = \nabla f_j^*$ for $m \equiv (i, j)$. But unless this equality holds for *all* $(i, j) \in \mathcal{E}$ (i.e., unless the minimum has been attained), λ will continue to change, and the ∇f_i^* will differ. This prevents us from easily designing a scenario in the decentralized setting where SGS-CD attains the speedup upper bound with respect to SU-CD. Nevertheless, in Figure 3.2 we show examples where (i) *the speedup increases linearly with* $N_{\max}$ for the decentralized setting, and (ii) *the speedup matches* $N_{\max}$ for the parallel distributed setting.

3.6.2 Dual-unfriendly functions and relation to dual ascent

The exposition that we have adopted up to this point may suggest that in order to run the set-wise CD methods presented here, one should be able to compute the Fenchel conjugates f_i^* for $i \in [n]$. Computing these functions may be tedious, and in some cases, like the logistic regression example presented in the next section, simply impossible.

However, we remark that the dual coordinate algorithm presented here is equivalent to the dual decomposition method (Section 2.2 in [100]) and therefore the gradients ∇f_i^* can be directly computed by minimizing the per-node Lagrangian (see also Proposition 11.3 in [101])

$$\nabla f_i^*(u_i^T A\lambda) = \arg\min_{\theta_i} \left[f_i(\theta_i) + \sum_{\ell \in \mathcal{S}_i} A_{i\ell}\lambda_\ell \theta_i \right]. \tag{3.37}$$

Therefore, to apply the algorithms presented here we do *not* need to be able to compute the Fenchel conjugates f_i^*, as long as we can solve (3.37) analytically or numerically to a high precision. This is what we do in our experiments of Section 3.7 for the logistic regression problem.

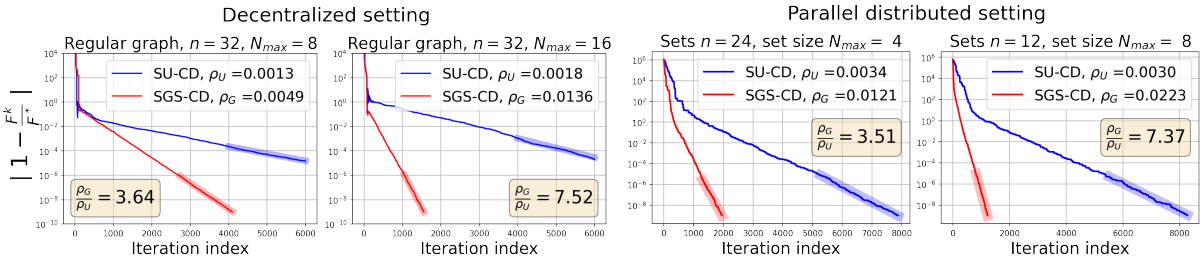

Figure 3.2: Comparison of the convergence rates of SU-CD and SGS-CD for quadratic problems in two settings: decentralized optimization over a network (left plots), and parallel distributed computation with parameter server (right plots).

3.6.3 Case $d > 1$

To extend the proofs above for $d > 1$, the block arrays Λ and U_i should be used instead of A and u_i, and the selector matrices T_i should be redefined in the same way (i.e., by making a Kronecker product with the identity). Then, all the operations that in the proofs above are applied *per entry* (scalar coordinate) of the vector λ, should now be applied to *the magnitude* of each vector (or "block" [46]) coordinate $\lambda_\ell \in \mathbb{R}^d$ of $\lambda \in \mathbb{R}^{Ed}$. Also, since $\nabla_m F \in \mathbb{R}^d$, in this case the GS rule becomes $\operatorname{argmax}_{m \in \mathcal{S}_i} \|\nabla_m F(\lambda)\|_2^2$ (and the GSL rule is modified analogously).

3.7 Numerical results

In this section, we test the algorithms proposed in numerical simulations and analyze their performance in a range of different scenarios. In all cases, we used (3.37) to compute the ∇f_i^* needed in (3.6). For quadratic and linear least squares problems (3.37) has a closed-form expression, while for logistic regression we used the SciPy module for the optimization [102].

3.7.1 SU-CD vs SGS-CD: speedup increase with $N_{\max}$

Figure 3.2 shows an example in the decentralized setting where the speedup of SGS-CD compared to SU-CD increases linearly with $N_{\max}$ (left plots), and an example in the parallel distributed setting where SGS-CD achieves the maximum speedup of $N_{\max}$ (right plots).

For the decentralized setting, we created two regular graphs of $n = 32$ nodes and degrees $N_{\max} = 8$ and 12, respectively. The local functions were $f_i(\theta) = \theta^T c I_d \theta$ with $d = 5$, and the constant c being much larger for one node than all others. This choice gave a few edges with smoothness constants much smaller than the rest, maximizing the chances to observe the advantages of SGS-CD versus SU-CD (see also the discussion in Section 4.1 of [45]).

For the parallel distributed setting, we created a problem that was separable per-coordinate,

and we tried to recreate the conditions described in Section 3.6.1 to approximate the $N_{\max}$ gain. We chose $F(x) = x^T \text{diag}(a_1, ..., a_d)x$ with $d = 48$ and $a_i \sim \mathcal{N}(10, 3)$ $\forall i$. We then created n sets of $N_{\max}$ coordinates such that each coordinate belonged to exactly two sets, and simulated two different distributions of the $d = 48$ coordinates: one with $n = 24$ sets of $N_{\max} = 4$ coordinates, and another with $n = 12$ sets of $N_{\max} = 8$ coordinates. Following the reasoning in Sec. 3.6.1, we set the initial value of $(N_{\max} - 1)$ coordinates in each set to $x_m^0 = 1$ (close to the optimal value $x_m^* = 0$), and the one remaining to $x_\ell^0 = 100$ (far away from $x_\ell^* = 0$).

In all plots of Fig. 3.2 we used the portion of the curves highlighted with thicker lines to estimate the suboptimality reduction factor $(1 - \rho)$, and called ρ_U and ρ_G the rates of SU-CD and SGS-CD, respectively. In all cases we see that $1 \leq \frac{\rho_G}{\rho_U} \leq N_{\max}$, as predicted by Theorem 13. We additionally observe that this ratio increases approximately in the same proportion as $N_{\max}$ for the decentralized setting, and is approximately equal to $N_{\max}$ in the parallel distributed.

3.7.2 Number of iterations vs communication complexity

Figure 3.3 shows the performance of all algorithms proposed for the linear least squares problem

$$f_i(\theta) = \frac{1}{M}\|X_i\theta - Y_i\|_2^2, \; X_i \in \mathbb{R}^{M\times d}, \; Y_i \in \mathbb{R}^M,$$

in two random graphs of $n = 32$ nodes and link probabilities of 0.1 (left plots) and 0.5 (right plots), respectively. The data was generated with the model of [35], $d = 5, M = 30$, and the Y values were additionally multiplied by the index of the corresponding node to have non-iid data between the nodes. Here we do not only show the convergence of the algorithms in terms of the number of iterations (top plots) but also in terms of the number of vectors in $\mathbb{R}^d$ transmitted through the network for each suboptimality value computed. Table 3.2 shows the communication complexity of each algorithm in these terms.

Table 3.2: Communication complexity of each algorithm: number of vectors in $\mathbb{R}^d$ transmitted in one iteration for an arbitrary activated node i. Variable I indicates the number of iterations inside the do-while loop in Alg. 4.

SU-CD	2	SGS-CD	$N_i + 1$
SL-CD	2	SGSL-CD	$N_i + 1$
SeL-CD	$2 + 2I$	SGSeL-CD	$N_i + 1 + 2I$

In terms of the number of iterations, we confirm the conclusions of all our corollaries, namely (i) SGS-CD converges faster than SU-CD (Corollary 3), (ii) SL-CD converges faster than SU-CD (Corollary 4), and (iii) SGSL-CD converges faster than both SL-CD and SGS-CD (Corollary 5). Whether the versions with estimated Lipschitz constants SeL-CD and SGSeL-CD are faster than their counterparts with exact Lipschitz knowledge SL-CD and SGSL-CD depends on the problem instance. We discuss this further in Section 3.8. As already observed in Fig. 3.2, the speedup of the algorithms applying either the GS or the GSL rule increases radically as the graph becomes more connected.

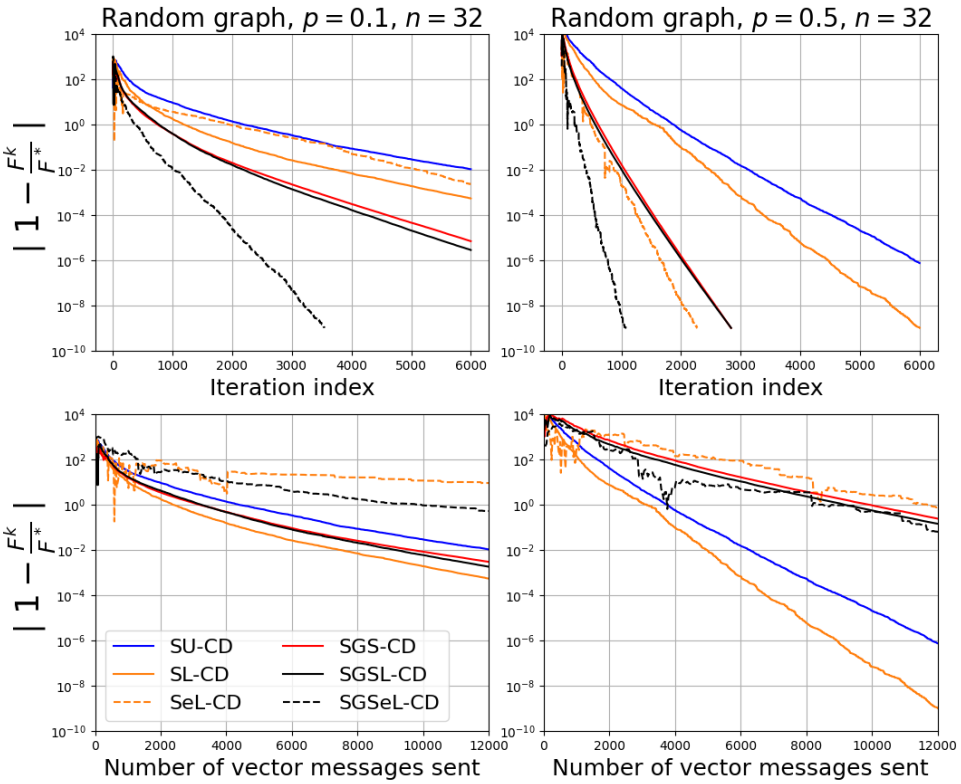

Figure 3.3: Performance of the algorithms presented in a linear least squares problem and two random graphs with different numbers of edges (left and right columns). The top plots show convergence in terms of the number of iterations and the bottom plots, in terms of the number of vectors in $\mathbb{R}^d$ transmitted.

The plots in terms of the number of vectors transmitted provide a complementary point of view of the relative performance of all these algorithms. We observe that for a sparsely connected graph (bottom-left plot in Fig. 3.3), SGS-CD and SGSL-CD may still achieve lower suboptimality than SU-CD for the same number of transmissions, but they are already outperformed by SL-CD. Algorithms SeL-CD and SGSeL-CD are the slowest when plotted against number of transmissions, since they have the additional overhead of estimating the smoothness constants. The gap between SU-CD and SL-CD, which are the algorithms with the lowest number of vector transmissions per iteration (see Table 3.2) and the rest of the algorithms becomes larger (in favor of the former) as the graph becomes more connected (bottom-left plot in Fig. 3.3). While it is natural that plotting the suboptimality reduction versus the number of vector transmissions benefits the algorithms using randomized neighbor selection (and no smoothness estimation), this does not necessarily mean that they will converge faster in a real system in terms of wall-clock time. We discuss this further in Section 3.8.

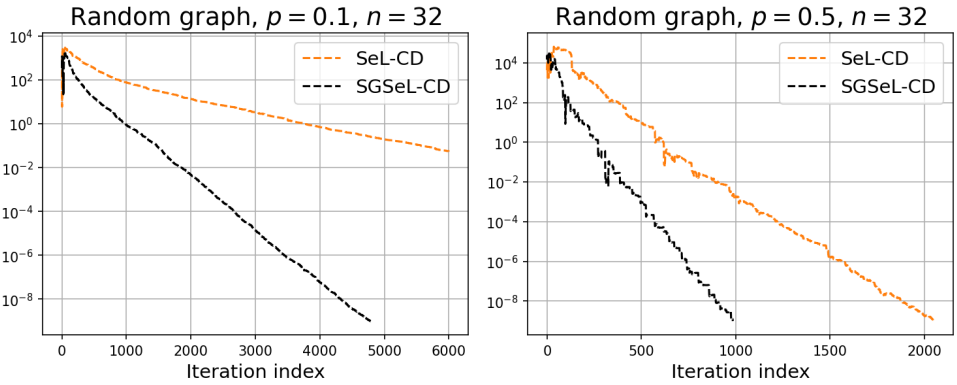

Figure 3.4: Convergence of SeL-CD and SGSeL-CD in a logistic regression problem.

3.7.3 A dual-unfriendly problem with no L knowledge

Figure 3.4 shows the convergence of SeL-CD and SGSeL-CD in the logistic regression problem

$$f_i(\theta) = \frac{1}{M}\sum_{j=1}^{M} \log\left(1 + \exp(-(Y_i)_j \cdot ((X_i)_j)^T \theta_i)\right) + c\|\theta_i\|_2^2$$

where $(Y_i)_j$ and $(X_i)_j$ are the j-th component and the j-th row of arrays Y_i and X_i, respectively. We ran the simulation for the same graphs and parameter choices used for the experiments in Section 3.7.2. In this case, we cannot compute analytically the optimal value of (3.37), so we did the optimization in (3.37) using the SciPy module. For the same reason, we do not know the true coordinate Lipschitz values L_ℓ, so we test only the algorithms using estimated constants.

As in the previous examples, both algorithms converge linearly, and SGSeL-CD is faster than SeL-CD. We may remark however that in this example the gap between the two algorithms does not increase with the graph connectivity, as observed with the algorithms that do not use estimated Lipschitz constants. Indeed, we observed that when the L_ℓ are estimated, the gap between SGSeL-CD and SeL-CD may or may not increase with the connectivity of the graph. We attribute this effect to the fact that the performance of the algorithms depends very much on how close to optimal the estimated Lipschitz constants are, and therefore, some instances that allow for a better fit of the true constants using Alg. 4 have advantage over others whose true constants cannot be well approximated. In the case of Fig. 3.4, we conjecture that the setting of the left plot allowed for particularly good estimation of the true Lipschitz constants by SeL-CD. In Section 3.8 we discuss in more detail how to improve the estimation of Alg. 4, and the associated costs of this improvement.

3.8 Discussion and conclusion

We have presented the class of *set-wise CD* optimization algorithms, where in a multi-agent system workers are allowed to modify only a subset of the total number of coordi-

nates at each iteration. These algorithms are suitable for (dual) asynchronous decentralized optimization and (primal) distributed parallel optimization.

We studied the convergence of a number of set-wise CD variants: random uniform and Gauss-Southwell set-wise coordinate selection (SU-CD and SGS-CD), and their Lipschitz-informed versions (SL-CD and SGSL-CD). We showed linear convergence for all variants for smooth and strongly convex functions f_i, which required developing a new methodology that extends previous results on CD methods.

In particular, we proved that in expectation, when convergence is measured *in terms of number of iterations*, both SGS-CD and SL-CD are faster than SU-CD, and SGSL-CD is the fastest of them all. However, running one iteration of each algorithm requires different amount of computation and communication (see Table 3.2). When the convergence is measured *in terms of number of vectors transmitted* through the network, the random algorithms become better than the ones based on the GS rule as the connectivity of the network increases. However, neither the performance measured against number of iterations nor of vectors transmitted is sufficient to decide which will perform the fastest in a real setup. Since all algorithms are asynchronous and can modify multiple coordinates simultaneously, in a real scenario many iterations and vector transmissions will occur at the same time, and the actual wall-clock time of the algorithm will depend on the network connectivity and where the bottlenecks of the system are (e.g. low link capacities, presence of stragglers).

Lastly, we proposed the methods SeL-CD and SGSeL-CD, which run respectively SL-CD and SGSL-CD but with online coordinate Lipschitz constant estimation for the cases where these values cannot be easily obtained. While these estimations can achieve remarkably good performance in terms of number of iterations (top plots in Fig. 3.3) they come with a greater penalty in number of vectors transmitted, which shows clearly in the bottom plots of Fig. 3.3. In cases where there is no alternative but to use estimated Lipschitz constants, as in our logistic regression example of Section 3.7.3, and if the communicaion constraints allow it, we could design an algorithm better than Alg. 4 that, instead of doubling $\widehat{L}_\ell$ at each time (line 5), runs a bisection search to approximate the true L_ℓ as much as possible. This would, of course, increase the communication cost even further, and once again, whether this penalty is worth paying is an engineering decision that will depend on the system setup and its constraints.

Chapter 4

Peer-to-peer aided federated learning

4.1 Introduction

Federated learning (FL) is a recent machine learning framework that allows multiple agents, each of them with their own dataset, to train a model collaboratively without sharing their data [47–49, 103]. The *federated* setting assumes that all agents are connected to a server that can communicate with each of them and that is in charge of aggregating the agents' updates to obtain the global model. This is similar to *parallel distributed* (PD) model training [96, 104–106], with one crucial difference: in the latter, the agents send *gradients* to the central server to update the parameter value with a *gradient step*, while in FL the agents send their *own local parameters* for the server to *average* them. This has an impact on the communication frequency required by each framework: in PD, one round of communication between (usually all) the agents and the server has to happen every time a (mini-batch) stochastic gradient descent (SGD) step is taken at the nodes, while in FL (i) multiple SGD updates can happen before a new server communication round takes place (which in FL literature are usually called *local updates*), and (ii) not all devices need to engage in the server communication round (which is known as *partial participation*). This makes FL a much more suitable option for settings with a large number of agents and a limited communication bandwidth with the server.

In contrast to the approaches described above, the *decentralized* setting does not rely on a central server for the aggregation of the nodes' updates. Instead, it assumes that the agents are interconnected in a network and each of them can exchange optimization values (either parameters or gradients, depending on the algorithm) with its direct neighbors [20, 24, 26, 28, 29, 35, 38]. In the decentralized setting, every node performs an averaging step of all its neighbors' received values before taking a new gradient step. Algorithms for this setting are designed such that the local parameters of all nodes converge to the global minimizer, while in FL it is the central server who keeps track of the most recent parameter value and broadcasts it to all agents every once in a while.

The attractive feature of FL of allowing to have server communication rounds every once in a while comes at a cost: the more infrequent the server communication rounds are (i.e., the more local updates are performed at the agents), the slower is the convergence [50,107]. For this reason, in this chapter we propose to exploit inter-agent communication to reduce the negative impact of infrequent server communication rounds. Given that (i) each agent is expected to have much fewer neighbors than the total number of agents, and (ii) short-range inter-agent communications allow for spectrum reuse, agents can communicate much more often between them than with the server [55, 56].

We propose FedDec, an FL algorithm where the agents can exchange and average their current parameters with those of their neighbors in between the local SGD steps. We show that this modification reduces the dependence of the convergence bound on the number of local SGD steps H from $O(H^2)$ [50, 54] to $O(H)$ (Theorem 16). Furthermore, we show that, in our analysis, the extra H factor is replaced by a value α that depends on the spectrum of the graph defining the inter-agent communication. Since the value of α quickly decreases as the network becomes more connected, our result indicates that for mildly connected networks H can be increased without severely hurting convergence speed (or conversely, for fixed H, FedDec will be faster than FedAvg [47], its counterpart without inter-agent communication).

Peer-to-peer communication within FL has been considered a few times in the past [57–61]. These works either analyze very general distributed settings that have FL with inter-agent communication as a particular case [60, 61], or show that FL with inter-agent communication converges at the same rate as standard FL, and outperforms it in simulations [57–59]. However, none of them has characterized analytically how inter-agent communication reduces the impact of local updates on convergence, and in particular, how this reduction depends on the inter-agent connectivity.

Our contributions can be summarized as follows:

- We introduce FedDec, an FL algorithm where the agents can average their parameters with those of their neighbors in between the SGD steps. Our model accounts for failures in the inter-agent communication links, so that only a few (or even none) of the parameters of a node's neighbors may be averaged at some iterations.

- We prove that, for non-iid data, partial device participation, and smooth and strongly convex objectives, FedDec converges at the $O(1/T)$ rate (where T is the total number of iterations executed) of FL algorithms that do not account for inter-agent communication [50, 54], but improves the dependence on the number of local updates H from $O(H^2)$ to $O(H)$.

- Furthermore, we show that the improved term is multiplied by a quantity that depends on the spectrum of the inter-agent communication network, and which quickly vanishes the more connected the network is.

- We support our theoretical findings with numerical simulations, where we confirm that the performance of FedDec with respect to FedAvg [47] increases with both H and the connectivity of the network.

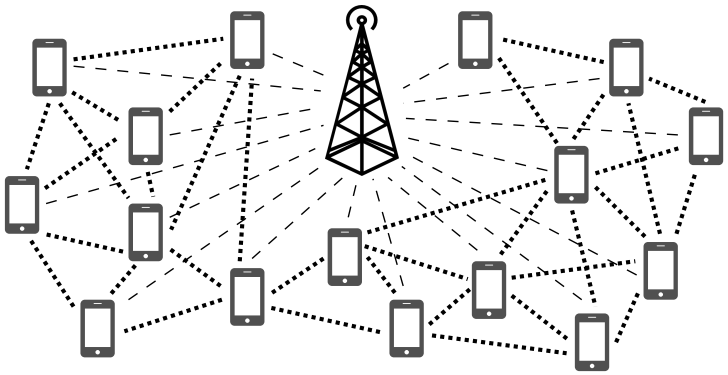

Figure 4.1: The FedDec setting. Peer-to-peer communication links are shown in dotted lines (high-bandwidth links). Communication links between the agents and the server are shown in dashed lines (low-bandwidth links).

4.2 System model and the FedDec algorithm

We consider a system where n agents can exchange messages with a central server and also with some other nearby agents. We assume that the inter-agent communication links may fail at some iterations (e.g. due to outage), but when all links are active the agents form a connected network (see Figure 4.1). Each node $i \in [n]$ has a local cost $F_i : \mathbb{R}^d \to \mathbb{R}$,

$$F_i(\mathbf{z}) := \mathbb{E}_{\psi_i \sim \mathcal{D}_i} F_i(\mathbf{z}, \psi_i),$$

where $\mathcal{D}_i$ can be an underlying local data distribution from where new samples (or mini-batches) are drawn each time an SGD step is taken, or the uniform distribution over a static dataset. Note that the $\mathcal{D}_i$ can be different at each node. The objective of the nodes and the server is to find the minimizer

$$\mathbf{z}^* := \operatorname*{argmin}_{\mathbf{z} \in \mathbb{R}^d} f(\mathbf{z}), \qquad f(\mathbf{z}) := \frac{1}{n} \sum_{i=1}^{n} F_i(\mathbf{z})$$

under the constraints that nodes can only communicate with their direct neighbors (high-bandwidth links) and every once in a while they can get a request from the server to send their current parameter values (low-bandwidth links). In wireless settings, these capacities are imposed by the shared nature of the cellular medium. While the communication with the server is constrained by the bandwidth available, device-to-device communications in the short range allows for spectrum reuse, and thus for higher throughput [55,56].

At each server communication round, the server samples the devices uniformly[1] at random with replacement to form an index pool $\mathcal{S}_t$ of devices that it will poll during that round. We assume $|\mathcal{S}_t| = K \; \forall t$. It then averages the parameters of all $j \in \mathcal{S}_t$ and broadcasts the new value to *all* nodes in the network. Due to the limited bandwidth, we assume partial participation, i.e. $K \ll n$. We assume that the server aggregation rounds happen every H local updates, and we call $\mathcal{H} = \{t : t \text{ modulo } H = 0\}$ the set of those times.

One local update of FedDec for a node i consists on (i) taking an SGD step, (ii) for all

[1] Our analysis is readily extendable to the case where the server samples with non-uniform probabilities $\{p_i\}_{i=1}^n$, in which case the cost becomes $f(\mathbf{z}) = \sum_{i=1}^n p_i F_i(\mathbf{z})$ and the term $\frac{\bar{\sigma}^2}{n}$ in Theorem 16 becomes $\sum_{i=1}^n p_i^2 \sigma_i^2$.

Algorithm 5 Peer-to-peer aided FL: FedDec algorithm

Initialize $\mathbf{z}_i^1 = \mathbf{z}^1 \; \forall i \in [n]$ and let $\eta_t = \frac{2}{\mu(t+\gamma)}$.
for $t = 1, \dots, T$ all agents $i \in [n]$ **do**
  Sample mixing matrix $W^t \sim \mathcal{W}$
  Sample mini-batch ξ_i^t and compute $\nabla F_i(\mathbf{z}_i^t, \xi_i^t)$
  Update local parameter $\mathbf{x}_i^{t+\frac{1}{2}} = \mathbf{z}_i^t - \eta_t \nabla F_i(\mathbf{z}_i^t, \xi_i^t)$
  Average with neighbors $\mathbf{x}_i^{t+1} = \sum_{j=1}^n W_{ij}^t \mathbf{x}_j^{t+\frac{1}{2}}$
  If $t+1 \in \mathcal{H}$ **then**
    Server samples $\mathcal{S}_t = \{j_\ell : j_\ell \sim \mathcal{U}([n])\}_{\ell=1}^K$
    it computes $\mathbf{z}^{t+1} = \frac{1}{K} \sum_{\ell=1}^K \mathbf{x}_{j_\ell}^{t+1}$
    and broadcasts so that $\mathbf{z}_i^{t+1} = \mathbf{z}^{t+1} \; \forall i \in [n]$
  otherwise
    $\mathbf{z}_i^{t+1} = \mathbf{x}_i^{t+1}$
end for
Output $\mathbf{z}^{T+1}$

active links (i.e. $\forall j \colon W_{ij}^t > 0$), exchanging the new parameter value with its neighbors, and (iii) combining all new values (including its own) with weights W_{ij}^t to form the new iterate. We call this algorithm FedDec, and the precise steps are shown in Algorithm 5.

With slight abuse of notation, the stochastic gradient of a node i computed on a mini-batch $\xi_i = \{\psi_i^j : \psi_i^j \sim \mathcal{D}_i\}_{j=1}^m$ of size m is given by $\nabla F_i(\mathbf{z}_i, \xi_i) = \frac{1}{m} \sum_{j=1}^m \nabla F_i(\mathbf{z}_i, \psi_i^j)$. For simplicity, we will assume that the number of iterations T satisfies (T modulo H) $= 0$, so that the outputted value in Alg. 5 is the current parameter at all nodes. We will also take the following assumptions, which are standard in the literature [29, 50, 54, 57–59].

Assumption 1. We assume the following $\forall F_i(\mathbf{z}), i \in [n]$:

1) **L-smoothness and μ-strong convexity:**

$$F_i(\mathbf{y}) \leq F_i(\mathbf{x}) + \langle \nabla F_i(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + (L/2)\|\mathbf{y} - \mathbf{x}\|_2^2, \tag{4.1}$$

$$F_i(\mathbf{y}) \geq F_i(\mathbf{x}) + \langle \nabla F_i(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + (\mu/2)\|\mathbf{y} - \mathbf{x}\|_2^2. \tag{4.2}$$

2) **Bounded variance of the local gradients:**

$$\mathbb{E}\|\nabla F_i(\mathbf{x}, \xi_i) - \nabla F_i(\mathbf{x})\|_2^2 \leq \sigma_i^2.$$

3) **Bounded energy of the local gradients:**

$$\mathbb{E}\|\nabla F_i(\mathbf{x}, \xi_i)\|_2^2 \leq G^2 \text{ for } i \in [n]. \tag{4.3}$$

We remark that (4.2) implies $\|\nabla F_i(\mathbf{z}_i)\|_2 \geq \mu \|\mathbf{z}_i - \mathbf{z}_i^*\|_2$ (see definition of $\mathbf{z}_i^*$ below). Therefore, in order to satisfy (4.3) we must additionally assume that the parameter iterates $\mathbf{z}_i$ belong to a bounded set throughout the iterations.

Note that L-smoothness implies

$$\|\nabla F_i(\mathbf{x}) - \nabla F_i(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|_2, \tag{4.4}$$

$$\|\nabla f(\mathbf{z})\|_2^2 = \|\nabla f(\mathbf{z}) - \nabla f(\mathbf{z}^*)\|_2^2 \leq 2L(f(\mathbf{z}) - f(\mathbf{z}^*)). \tag{4.5}$$

Furthermore, the local gradients' bounded variance implies

$$\mathbb{E}\left\|\frac{1}{n}\sum_{i=1}^{n}\big(\nabla F_i(\mathbf{z}_i^t) - \nabla F_i(\mathbf{z}_i^t, \xi_i^t)\big)\right\|_2^2 \leq \frac{\overline{\sigma}^2}{n}, \tag{4.6}$$

with $\overline{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}\sigma_i^2$.

We quantify the degree of heterogeneity (or non-iidness) between the local functions through the quantity

$$\Gamma = \frac{1}{n}\sum_{i=1}^{n}(F_i(\mathbf{z}^*) - F_i(\mathbf{z}_i^*)), \text{ where } \mathbf{z}_i^* = \operatorname*{argmin}_{\mathbf{z}} F_i(\mathbf{z}).$$

FedDec uses two parameters to track the updates: $\mathbf{z}_i$, the parameters just before the SGD and neighbor averaging steps, and $\mathbf{x}_i$, the parameters just after those steps. These values may be equal or not depending on t and whether it is a server communication round (see Alg. 5). We define

$$\overline{\mathbf{x}}^t = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i^t, \qquad \overline{\mathbf{z}}^t = \frac{1}{n}\sum_{i=1}^{n}\mathbf{z}_i^t,$$

which will be useful in the analysis. Note that $\overline{\mathbf{z}}^t = \overline{\mathbf{x}}^t$ only when $t \notin \mathcal{H}$. Otherwise, if $t \in \mathcal{H}$, the equality holds only in expectation:

$$\mathbb{E}_{\mathcal{S}_t}\overline{\mathbf{z}}^t = \mathbb{E}_{\mathcal{S}_t}\frac{1}{n}\sum_{i=1}^{n}\mathbf{z}_i^t = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{K}\sum_{\ell=1}^{K}\mathbb{E}_{\mathcal{S}_t}\mathbf{x}_{j_\ell}^t = \overline{\mathbf{x}}^t. \tag{4.7}$$

Lastly, we assume the following about the $W^t = \{W_{ij}^t\}$.

Assumption 2. The averaging matrices $W^t \in \mathbb{R}^{n\times n}$ are iid random variables drawn from a distribution $\mathcal{W}$ of matrices that (i) are symmetric, (ii) are doubly stochastic, and (iii) have $W_{ij}^t \geq 0$ if agents i and j are connected and $W_{ij}^t = 0$ otherwise. Note that this implies that $\forall W \in \mathcal{W} : W\mathbf{1} = \mathbf{1}$, $\mathbf{1}^T W = \mathbf{1}^T$. Additionally, we require that the eigenvalues of $\mathbb{E}_W\left[WW^T\right]$ satisfy $1 = \lambda_1 > |\lambda_2| \geq \ldots \geq |\lambda_n|$.

In the next section, we prove that FedDec converges as $O(1/T)$, similarly to other FL algorithms taking the same assumptions, but it reduces the negative impact of local updates by replacing an H^2 factor [50,54], with $H\alpha$, where α is a quantity that decreases quickly as the inter-agent communication network becomes more connected.

4.3 Convergence analysis

The following theorem establishes the convergence rate of FedDec and constitutes our main result.

Theorem 16. *Under Assumptions 1, 2, and for diminishing stepsize* $\eta_t = \frac{2}{\mu(\gamma+t)}$, *FedDec in Algorithm 5 converges as*

$$\mathbb{E}[f(\overline{\mathbf{z}}^t)] - f(\mathbf{z}^*) \leq \frac{L}{\gamma+t}\left(\frac{2B}{\mu^2} + \frac{(\gamma+1)}{2}\left\|\mathbf{z}^1 - \mathbf{z}^*\right\|_2^2\right)$$

where

$$\gamma = \max\{8(L/\mu) - 1, H\}$$
$$B = (4/K + 8)\alpha H G^2 + 6L\Gamma + \overline{\sigma}^2/n$$
$$\alpha = |\widehat{\lambda}_2| \,/\, (1 - |\widehat{\lambda}_2|)$$

and $|\widehat{\lambda}_2| = |\lambda_2(\mathbb{E}_W[WW^T])|$.

The theorem shows that factors like the energy and the variance of the local gradients, the heterogeneity of the local functions, and the distance of the starting point to the optimum all slow down convergence, which are known facts. However, this bound also shows how inter-agent communication partially mitigates the negative impact of local updates: the term where H appears decreases with α, and therefore decreases very fast with $|\widehat{\lambda}_2|$ (see Figure 4.2).

Note that if all inter-agent communication links are assumed to be always active, then $W^t = W$ is a fixed matrix and $|\widehat{\lambda}_2| = |\lambda_2|^2$. For any given heuristic to construct W (e.g. based on the Laplacian of the graph [108]), the value of $|\lambda_2|$ is, in general, lower the more connected the network is (see Table 4.1 in Section 4.4). Therefore, the more densely connected the network is, the faster FedDec is expected to converge. In fact, the averaging weights W_{ij} can be designed in order to minimize $|\lambda_2|$ (and thus maximize the speedup from inter-agent communication) using eigenvalue optimization techniques [109].

Comparing the bound of Theorem 16 with that of Theorem 2 in [50], obtained for the same setting but without allowing inter-agent communication, we note that the dependence of the first term in B on the number of local iterations H drops from $O(H^2)$ in [50] to $O(H)$ in our theorem. This suggests that *the peer-to-peer communication of FedDec reduces the impact of the infrequent communication rounds with the server*, and thus its convergence should be less affected than that of FedAvg as H increases. We verify this behavior in our simulations in Section 4.4.

To prove the theorem, we will need to bound the quantity $\|\overline{\mathbf{z}}^t - \mathbf{z}^*\|_2^2$. For this we will decompose the term and bound $\|\overline{\mathbf{z}}^t - \overline{\mathbf{x}}^t\|_2^2$ and $\|\overline{\mathbf{x}}^t - \mathbf{z}^*\|_2^2$ separately. The following lemmas present these intermediate results, and we prove Theorem 16 at the end of the section. The proofs of the lemmas are given in Appendix B.2.

Lemma 17. *For FedDec with stepsize* $\eta_t \leq \frac{1}{4L}$ *it holds*

$$\mathbb{E}\left\|\overline{\mathbf{x}}^{t+1} - \mathbf{z}^*\right\|_2^2 \leq (1 - \mu\eta_t)\mathbb{E}\left\|\overline{\mathbf{z}}^t - \mathbf{z}^*\right\|_2^2 + \frac{2}{n}\mathbb{E}\sum_{i=1}^n \left\|\mathbf{z}_i^t - \overline{\mathbf{z}}^t\right\|_2^2 + 6L\eta_t^2\Gamma + \eta_t^2\frac{\overline{\sigma}^2}{n}.$$

Lemma 17 bounds the one-step progress of the algorithm *before* a potential server aggregation round.

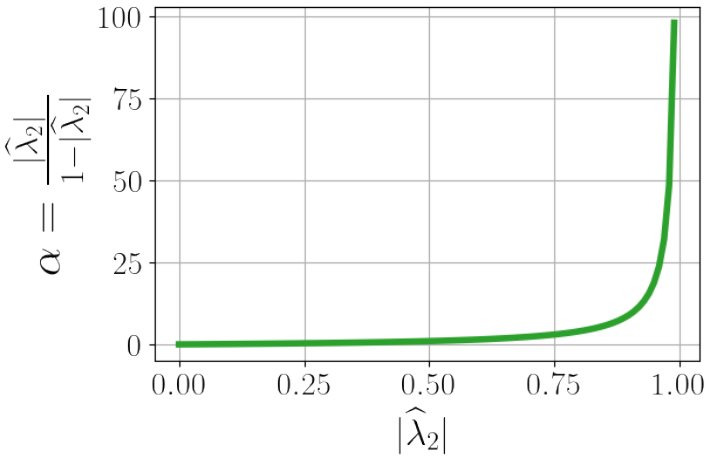

Figure 4.2: Dependence of α with $|\widehat{\lambda}_2|$. For networks moderately connected (see Section 4.4) we can expect that $\alpha \ll H$, and thus also that FedDec will be faster than FedAvg.

Lemma 18. *For stepsizes satisfying $\eta_t \leq 2\eta^{t+H}$, it holds that*

$$\mathbb{E}\sum_{i=1}^{n}\left\|\mathbf{z}_i^t - \overline{\mathbf{z}}^t\right\|_2^2 \leq \eta_t^2 4\alpha H n G^2$$

with $\alpha = \frac{|\widehat{\lambda}_2|}{1-|\widehat{\lambda}_2|}$.

Lemma 18 bounds the divergence of the local parameters to their average, which increases through the H iterations in between the server broadcasting rounds. It is in this process (which involves multiple neighbor averaging steps) where we see the impact of the connectivity of the graph.

As remarked in Section 4.2, $\overline{\mathbf{z}}^t = \overline{\mathbf{x}}^t \, \forall \, t \notin \mathcal{H}$. Otherwise, the equality holds only in expectation (eq. (4.7)). Lemma 19 bounds the variance of $\overline{\mathbf{z}}$ in the latter case.

Lemma 19. *For $t \in \mathcal{H}$ and stepsizes satisfying $\eta_t \leq 2\eta_{t+H}$*

$$\mathbb{E}\left\|\overline{\mathbf{x}}^t - \overline{\mathbf{z}}^t\right\|_2^2 \leq \frac{1}{K}\eta_t^2 4\alpha H G^2,$$

with α given in Lemma 18.

Lastly, we have the following lemma from [50].

Lemma 20. *Let a sequence Δ^t satisfy*

$$\Delta^{t+1} \leq (1-\mu\eta_t)\Delta^t + \eta_t^2 B \tag{4.8}$$

with $\mu, B > 0$. Then, for a diminishing stepsize $\eta_t = \frac{2}{\mu(\gamma+t)}$ with $\gamma > 0$, it holds that $\Delta_t \leq \frac{v}{\gamma+t}$, where $v = \max\{\frac{4B}{\mu^2}, (\gamma+1)\Delta_1\}$.

Proof. See proof of Theorem 1 in [50]. □

This bound establishes the parameter choices that allow the sequence Δ^t to converge. We have now all the tools necessary to prove the main theorem.

Proof of Theorem 16. We start by noting that

$$\begin{aligned}\mathbb{E}\left\|\overline{\mathbf{z}}^{t+1}-\mathbf{z}^*\right\|_2^2 &= \mathbb{E}\left\|\overline{\mathbf{z}}^{t+1}-\overline{\mathbf{x}}^{t+1}+\overline{\mathbf{x}}^{t+1}-\mathbf{z}^*\right\|_2^2 \\ &= \mathbb{E}\left\|\overline{\mathbf{z}}^{t+1}-\overline{\mathbf{x}}^{t+1}\right\|_2^2+\mathbb{E}\left\|\overline{\mathbf{x}}^{t+1}-\mathbf{z}^*\right\|_2^2+2\mathbb{E}\left\langle\overline{\mathbf{z}}^{t+1}-\overline{\mathbf{x}}^{t+1},\overline{\mathbf{x}}^{t+1}-\mathbf{z}^*\right\rangle.\end{aligned}$$

The last term becomes zero when taking expectation, since $\mathbb{E}_{\mathcal{S}_t}\overline{\mathbf{z}}^t=\overline{\mathbf{x}}^t$ (eq. (4.7)). The first term is zero when $t+1\notin\mathcal{H}$, and for all other iterations we can bound it using Lemma 19 (and the fact that $\eta_{t+1}^2<\eta_t^2$). We bound the second term using Lemma 17. We have then

$$\mathbb{E}\left\|\overline{\mathbf{z}}^{t+1}-\mathbf{z}^*\right\|_2^2\leq\frac{1}{K}\eta_t^2 4\alpha HG^2+(1-\mu\eta_t)\mathbb{E}\left\|\overline{\mathbf{z}}^t-\mathbf{z}^*\right\|_2^2+\frac{2}{n}\mathbb{E}\sum_{i=1}^n\left\|\mathbf{z}_i^t-\overline{\mathbf{z}}^t\right\|_2^2+6L\eta_t^2\Gamma+\eta_t^2\frac{\overline{\sigma}^2}{n}.$$

Using Lemma 18 to bound $\mathbb{E}\sum_{i=1}^n\|\mathbf{z}_i^t-\overline{\mathbf{z}}^t\|_2^2$ we get

$$\mathbb{E}\left\|\overline{\mathbf{z}}^{t+1}-\mathbf{z}^*\right\|_2^2\leq(1-\mu\eta_t)\mathbb{E}\left\|\overline{\mathbf{z}}^t-\mathbf{z}^*\right\|_2^2+\eta_t^2\left[\left(\frac{4}{K}+8\right)\alpha HG^2+6L\Gamma+\frac{\overline{\sigma}^2}{n}\right].$$

This has the form of (4.8) with $\Delta^t=\|\overline{\mathbf{z}}^t-\mathbf{z}^*\|_2^2$ and $B=\left(\frac{4}{K}+8\right)\alpha HG^2+6L\Gamma+\overline{\sigma}^2/n$, so applying Lemma 20:

$$\mathbb{E}\left\|\overline{\mathbf{z}}^t-\mathbf{z}^*\right\|_2^2\leq\frac{v}{\gamma+t}\leq\frac{1}{\gamma+t}\left(\frac{4B}{\mu^2}+(\gamma+1)\Delta_1\right).\tag{4.9}$$

Note that in order to ensure $\eta_t\leq\frac{1}{4L}$ (Lemma 17) and $\eta_t\leq 2\eta_{t+H}$ (Lemmas 18 and 19) we need to set $\gamma=\max\{8\frac{L}{\mu}-1,H\}$. Finally, using L-smoothness and $\nabla f(\mathbf{z}^*)=0$,

$$\mathbb{E}[f(\overline{\mathbf{z}}^t)]-f(\mathbf{z}^*)\leq\frac{L}{2}\mathbb{E}\left\|\overline{\mathbf{z}}^t-\mathbf{z}^*\right\|_2^2.$$

Using (4.9) in the inequality above gives the result. □

4.4 Numerical results

In this section we compare the performance of FedDec with that of FedAvg [47] in a problem with partial device participation and heterogeneous data.

We consider the linear regression problem

$$F_i(\mathbf{z})=\frac{1}{M}\|X_i\,\mathbf{z}-Y_i\|_2^2,\qquad i\in[n],$$

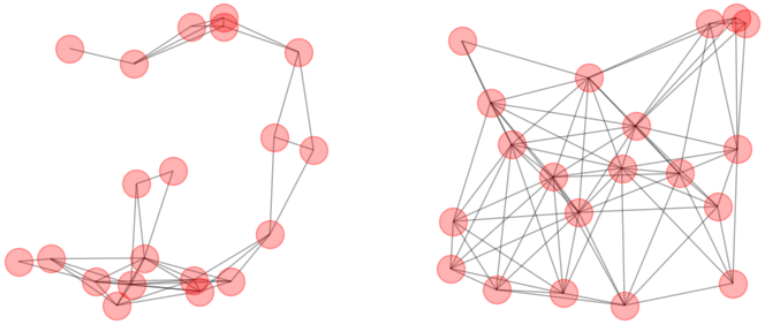

Figure 4.3: Graphs used in the simulations. **Left:** sparse graph with $r = 0.35$. **Right:** dense graph with $r = 0.5$.

with $X_i, Y_i \in \mathbb{R}^{M \times d}$, $M = 10$, and $d = 25$. For generating the regression data we follow a procedure similar to [35]: we set $[X_i]_j \sim \mathcal{N}(0, 0.25^2)$, $j \in [d]$ and $Y_i = c_i(v + \cos(v))$, where $v = X_i \mathbf{1}$ and $c_i = 2^i, i \in [n]$ is a factor that makes the data at each node significantly different from all others.

For the inter-agent communication, we generate geographic graphs of $n = 20$ nodes by taking n points distributed uniformly at random in a 1×1 square and joining with a link all pairs of points whose Euclidean distance is smaller than a radius r. We test our algorithms in two graphs with $r = 0.35$ and 0.5, respectively (see Figure 4.3). We run $T = 5000$ iterations with $K = 2$, $m = 1$, and $H = 10, 100$.

Figure 4.4 shows the convergence of FedDec and FedAvg for each graph in Fig. 4.3 and the two values of H. The stepsize was set to the value indicated in Theorem 16. The lines shown are the average of ten independent runs of the algorithms on the same problem instance.

Comparing the plots in Fig. 4.4 vertically (i.e., comparing the two graphs for the same H), we confirm that higher connectivity leads to larger gains of FedDec over FedAvg. This can be understood intuitively by noticing that a denser graph facilitates a faster spread of information. Since $|\lambda_2|$ correlates with the graph connectivity (see Table 4.1), in this case it is a good predictor of the convergence speed of FedDec. However, we note that it has been reported that connectivity, as measured by $|\lambda_2|$, seems to be predictive of the convergence speed of decentralized algorithms (in terms of number of iterations) only when the nodes have sufficiently different data [91], as is the case in our simulations. When the data is iid among the nodes, the *number of effective neighbors* seems to be a better predictor [110].

Comparing the plots in Fig. 4.4 horizontally (i.e., comparing the effect of changing H for a given graph), we verify that as H increases the convergence speed of FedAvg decreases more than that of FedDec. Therefore, FedDec allows for sparser server communication rounds without significantly sacrificing convergence speed, which is in accordance with Theorem 16.

One may wonder whether in practice α takes values much smaller than H so that the gains of FedDec can actually be observed. Assuming a fixed W, this question is equivalent to asking what are typical values for $|\lambda_2|^2$ (see Fig. 4.2). Table 4.1 shows this value for many

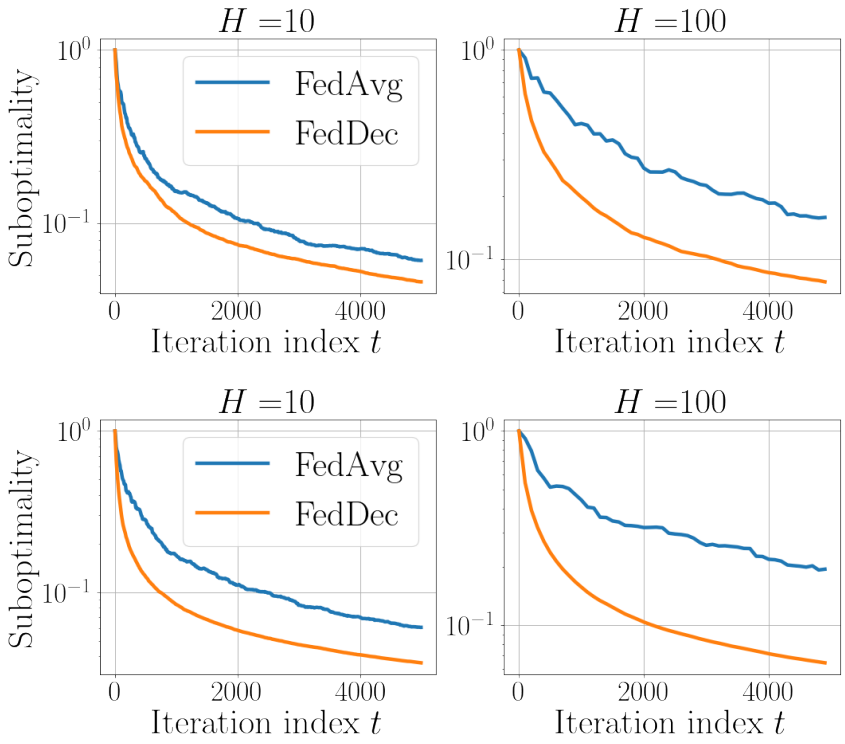

Figure 4.4: FedDec versus FedAvg for the sparse graph (top) and the dense graph (bottom). When the communication with the server is less frequent (larger H), the performance of FedAvg is more degraded than that of FedDec.

graphs, where W was constructed using the graph's Laplacian [108]. We computed these values for geographic graphs with different linking radii r and random (Erdős–Rényi) graphs with different link probabilities p, and in both cases, for different number of nodes. Geographic graphs are good models for wireless networks [111], while random graphs have the small-world property of other kinds of networks, such as the Internet [112]. The values of r and p are such that the networks in the same row and column under each graph type have approximately the same number of edges. The numbers shown are the average over 10 independent realizations, and the values corresponding to the graphs in Fig. 4.3 are shown in gray. We observe that in all cases $|\lambda_2|^2 < 0.9$, which implies $\alpha < 9$.

Table 4.1: Values of $|\lambda_2|^2$ for different graphs.

Geographic graph			
Connection radius	$n = 10$	$n = 20$	$n = 40$
$r = 0.35$	**0.78**	**0.87**	**0.83**
$r = 0.5$	**0.7**	**0.64**	**0.56**
$r = 0.65$	**0.41**	**0.33**	**0.34**
Random graph			
Link probability	$n = 10$	$n = 20$	$n = 40$
$p = 0.3$	**0.7**	**0.62**	**0.4**
$p = 0.5$	**0.42**	**0.29**	**0.17**
$p = 0.7$	**0.25**	**0.13**	**0.083**

Therefore, unless H is particularly small, FedDec is expected to be (potentially, much) faster than FedAvg. In particular, for random graphs, which have low network diameter, $|\lambda_2|^2$ decreases more abruptly as connectivity grows. This further indicates that for well-connected networks, inter-agent communication can make the first term of B in Theorem 16 become negligible.

4.5 Conclusion

We have presented FedDec, an algorithm that exploits inter-agent communication in FL settings by averaging the agents' parameters with those of their neighbors before each new local SGD update. We proved that this modification reduces the negative impact of local updates on convergence and that the magnitude of this reduction depends on the spectrum of the graph defining the communication network. This further indicates that the effect of local updates and partial device participation can become negligible if the communication network is well-connected.

This insight suggests that there exists a connectivity threshold where the server does not help convergence anymore. Furthermore, we conjecture that for sufficiently dense networks, server communication rounds might even hurt. Future directions include studying this threshold and other trade-offs of peer-to-peer aided FL.

On the other hand, if inter-agent communication does not form a connected network when all links are active, communication with the server is essential to have convergence. Interesting questions to tackle in this scenario include how often the server should communicate with each sub-network in order to converge, and how this value depends on the heterogeneity of data *between* subnetworks.

In any case, FedDec should be seen as an algorithm that *redistributes* the communication load in FL by reducing the agent-server communication and increasing (in fact, *introducing*, or making larger than zero) the inter-agent. We expect that in real systems this better utilization of the medium will lead to shorter wall-clock training time.

Chapter 5

Conclusion

In this thesis, we have proposed solutions for making better utilization of the resources of communication networks in the context of two timely applications: popular content delivery and distributed machine learning. As the number of devices connected to the Internet rapidly increases, it becomes imperative to design protocols and algorithms that make the best use of the resources available in order to cope with the demands without saturating the network. The techniques presented here have as the main goal to increase the performance of techniques designed for the two applications of interest either at the same or slightly higher communication cost.

In the context of popular content delivery, we have proposed to exploit recommendation systems to boost the performance of caching networks by designing jointly the contents to store in the cache and to recommend to the user (Chapter 2). Unlike previous work attempting to exploit recommendations to improve the performance of caching systems, our proposed algorithm has approximation guarantees to the jointly optimal solution. We confirmed in simulations that this algorithm outperforms others that take these decisions independently instead of jointly, in some cases with remarkable gains. By increasing the CHR exploiting recommendations, our algorithm reduces the network load by requesting fewer contents from the remote server than its competitors. Furthermore, this is achieved *without entailing any system setup changes*. In contrast, achieving a higher CHR with standard caching approaches would require increasing both the cache size and the number of files transmitted through the network. Our algorithm achieves the same result without these added communication and storage costs.

During the benchmarking of our proposed algorithm, we made the following interesting observation: while it always outperformed its competitors, the magnitude of the gains varied significantly with the dataset tested. This led us to investigate how the dataset structure and the system setup affected the performance of our algorithm with respect to a very simple heuristic based on popularity caching. We identified a few features of the dataset and system setup values that affected significantly the CHR gap between the two algorithms. Most notably, we identified *interactions* between pairs of features that could either enlarge or close the performance gap, such as skewed degree distribution and the number of recommendations, or cache size and popularity skewness. Based on these observations, we trained a classifier to predict when our proposed algorithm could

provide significant gains with respect to the baseline for a given system setup and dataset structure, with very good results. Such classifiers could, *on top* of the communication gains provided by the caching systems and the exploitation of recommendation engines, bring *computational gains*: the classifier could tell us whether we can expect significant performance gains by utilizing our proposed polynomial-time algorithm, or whether the system setup and dataset structure are such that the low-cost heuristic will do just as well.

In the context of distributed machine learning, we have studied two particular configurations, the decentralized and the federated, which differ in the communication scheme used by the agents. For the decentralized setting, where the agents can communicate between them and there is no central coordinator, we have proposed a family of algorithms that solve the distributed machine learning model problem through its dual, which can be run asynchronously and can be seen as a coordinate descent algorithm where only a small subset of all the coordinates can be modified at each iteration (Chapter 3). Thanks to this insight, we proposed adaptations of standard (single-machine) coordinate descent methods to our dual asynchronous decentralized algorithms.

However, the subtle difference of having only a subset of the total number of coordinates to choose from complicated the analysis substantially, and required defining particular norms for each algorithm in which to measure the function's strong convexity. It was also important to keep the bounds as tight as possible in order to compare the convergence rate of the multiple variants proposed. In particular, we showed that (i) exploiting either the knowledge of the coordinate Lipschitz constants or the Gauss-Southwell rule, which chooses to update the dual variable with the largest gradient from those accessible, attains faster per-iteration performance than random uniform sampling, and (ii) exploiting both simultaneously achieves the largest gains. However, using the GS rule (and also the Lipschitz constants, if they need to be estimated) requires more communication between the agents than random uniform sampling. However, in settings where the available resources can accommodate this increased communication complexity, the algorithms based on GS rule and/or coordinate Lipschitz exploitation have the potential to reach the target accuracy in less wall-clock time.

The federated setting, on the other hand, considers in principle that the agents can only communicate with a central server that once in a while requests their local parameters, averages them with those of (a subset of all) other agents, and broadcasts the result to all nodes. In this setting, the agent-to-server link constitutes the main communication bottleneck, which is the reason why it is usually assumed that agents communicate with the server only once in a while and only a few agents send their local parameters at each server communication round. For this reason, here we have proposed exploiting inter-agent communication in the context of federated learning to allow for more infrequent communication with the server without hurting convergence speed (Chapter 4). Since low-range communications allow for spectrum reuse, the agents can communicate more frequently between them than with the server. We show that allowing the parameter exchange and averaging that is proper of decentralized primal methods in between the local updates of federated learning reduces the dependence on the number of local updates H from $O(H^2)$ to $O(H)$, and this term vanishes as the network becomes more connected. This indicates that exploiting inter-agent communication can achieve the same convergence speed as standard federated learning with much sparser communication with the

server, alleviating therefore the communication bottleneck of this setting.

It is worth noting that the methods presented here can be combined with many recent proposals in the literature to further improve the performance of the system at the same communication cost or, equivalently, to lower the communication cost without degrading performance. In the context of network caching, examples of these techniques are data encoding and explicit energy optimization [113], and accounting for the age of information of the contents cached [114]. For distributed machine learning, examples are gradient compression techniques [29, 115] and variance reduction [49].

Overall, the reader should take the methods presented in this thesis as examples of how system performance can be improved by carefully designing better algorithms that entail minimal or no structural changes. In the best case, they will also serve as an inspiration and grounds for new and high-performing optimization methods over networks.

Appendix A

Proofs of Chapter 2

A.1 Proof of Lemma 3

According to Lemma 2, for set S, it holds that $f_k^*(S) = 1-(1-u_k^{(1)}(S))\cdot(1-u_k^{(2)}(S))\dots(1-u_k^{(N)}(S))$. Now, assume that we add some element i in S. Then, there exist two cases:

(a) If $u_{ki} \leq u_k^{(N)}(S)$, then $f_k^*(S \cup \{i\}) = f_k^*(S)$, the objective remains unchanged.

(b) If $u_{ki} > u_k^{(N)}(S)$, then

$$\begin{aligned} f_k^*(S \cup \{i\}) &= 1 - (1 - u_k^{(1)}(S)) \dots (1 - u_k^{(N-1)}(S)) \cdot (1 - u_{ki}) \\ &= 1 - (1 - f_k^*(S)) \frac{(1 - u_{ki})}{(1 - u_k^{(N)}(S))} \\ &\overset{(a)}{\geq} f_k^*(S) \left(1 - \frac{(1 - u_{ki})}{(1 - u_k^{(N)}(S))}\right) + f_k^*(S) \frac{(1 - u_{ki})}{(1 - u_k^{(N)}(S))} \\ &= f_k^*(S), \end{aligned}$$

where in (a) we used the fact that $f_k^*(S) \leq 1$.

A.2 Proof of Theorem 4

Consider the powerset $\mathcal{F}$ of $\{1, 2, \dots, K\}$, i.e. the set of all subsets of S. Assume that we add element i to some set $S \in \mathcal{F}$. We denote

$$\Delta f(S, i) = f_k^*(S \cup \{i\}) - f_k^*(S). \tag{A.1}$$

Then, we can consider the following cases:

$$\left(u_{ki} \leq u_k^{(N)}(S)\right) \Rightarrow \Delta f(S, i) = 0.$$

$$\left(u_{ki} > u_k^{(N)}(S))\right) \Rightarrow \Delta f(S,i) = (1 - u_k^{(1)}(S)) \dots (1 - u_k^{(N-1)}(S)) \cdot \left[u_{ki} - u_k^{(N)}(S)\right] \quad \text{(A.2)}$$

To show submodularity, we need to prove that $\Delta f(A,i) - \Delta f(B,i) \geq 0$, for any sets $A, B \in \mathcal{F}$, such that $A \subset B$, and any $i \in \{1, 2, \dots, K\}$. There are three separate cases to consider:

(Case 1) $\Delta f(A,i) = \Delta f(B,i) = 0$: Adding element i to either set A or B does not improve the objective (i.e. the new content i added to the cache has a value u_{ki} that is lower than the top N values of contents already in A (or B)).

(Case 2) $\Delta f(A,i) > 0, \Delta f(B,i) = 0$: Then, according to eq. (A.2) we have that

$$\Delta f(A,i) - \Delta f(B,i) = (1 - u_k^{(1)}(S)) \dots (1 - u_k^{(N-1)}(S)) \cdot [u_{ki} - u_k^{(N)}(S)]$$

which is strictly higher than 0.

(Case 3) $\Delta f(A,i) > 0, \Delta f(B,i) > 0$: In this case, it is easy to see that according to Lemma 3 the highest order statistics of sets A and B will coincide up to some order m, and will differ from order $m+1$ up to N. Then, we can write

$$\begin{aligned} \Delta f(A,i) = (1 - u_k^{(1)}(B)) \dots (1 - u_k^{(m)}(B))(1 - u_k^{(m+1)}(A)) \dots \\ \dots (1 - u_k^{(N-1)}(A)) \left[u_{ki} - u_k^{(N)}(A)\right] \quad \text{(A.3)} \\ \Delta f(B,i) = (1 - u_k^{(1)}(B)) \dots (1 - u_k^{(m)}(B))(1 - u_k^{(m+1)}(B)) \dots \\ \dots (1 - u_k^{(N-1)}(B)) \cdot \left[u_{ki} - u_k^{(N)}(B)\right] \quad \text{(A.4)} \end{aligned}$$

Let's denote the common term in the product as C, and as C_A and C_B the different terms in each product, respectively. Then,

$$\begin{aligned} \Delta f(A,i) - \Delta f(B,i) &= C \left[C_A \left(u_{ki} - u_k^{(N)}(A)\right) - C_B \left(u_{ki} - u_k^{(N)}(B)\right)\right] \\ &\geq C \cdot C_B \cdot \left[\left(u_{ki} - u_k^{(N)}(A)\right) - \left(u_{ki} - u_k^{(N)}(B)\right)\right] \quad \text{(A.5)} \\ &= C \cdot C_B \cdot \left(u_k^{(N)}(B) - u_k^{(N)}(A)\right) \geq 0. \quad \text{(A.6)} \end{aligned}$$

Eq. (A.5) follows from the fact that $f_k^*(S)$ is monotonically increasing and thus

$$f_k^*(B) \geq f_k^*(A) \Rightarrow 1 - C \cdot C_B \geq 1 - C \cdot C_A \Rightarrow C_A \geq C_B.$$

Eq. (A.6) also follows easily from Lemma 3, as all order statistics are monotonically increasing in the cardinality of the considered set.

Finally, the case $\Delta f(A,i) = 0, \Delta f(B,i) > 0$ cannot occur due to f_k^* being monotonically increasing in the cardinality of the chosen set of elements. This concludes the proof that $\Delta f(A,i) - \Delta f(B,i) \geq 0$ for all possible cases, and thus that f_k^* is submodular.

A.3 Proof of Corollary 2

We denote

$$f_{ki}(X, \mathbf{Y}^i) = \left[1 - \prod_{n=1}^{K} \left[\prod_{m=1}^{M} (1 - x_{nm} \cdot q_{im}) + \left(1 - \prod_{m=1}^{M} (1 - x_{nm} \cdot q_{im})\right)(1 - u_{kn}^i \cdot y_{kn}^i)\right]\right]$$

and $f_{ki}^*(X) = \max_{\mathbf{Y}^i} f_{ki}(X, \mathbf{Y}^i)$.

We can then repeat the argument of Lemma 1 to show that $F^*(X) = \sum_k \sum_i p_k^i f_{ki}^*(X)$, since we can maximize the terms in the sum for each k and i independently.

We denote now $S = \{(\ell, m), \ell \in \{1, K\}, m \in \{1, M\} : x_{\ell m} = 1\}$ the pairs (content, helper) such that content ℓ is stored in helper m. We will use the notation $f_{ki}^*(S)$ and $f_{ki}^*(X)$ interchangeably. Furthermore, let $S_i \subset S$ be the subset of pairs in S such that user i has access to a helper $m \in S$, i.e. $q_{im} = 1$. Note that since a user cannot get any benefit (cache hit) from a helper that they are not connected to, $f_{ki}^*(S) = f_{ki}^*(S_i)$, and furthermore

$$f_{ki}^*(S_i) = 1 - (1 - u_k^{(1)}(S_i)) \cdot (1 - u_k^{(2)}(S_i)) \dots (1 - u_k^{(N)}(S_i))$$

by the same arguments used in Lemma 2.

Therefore, we can decouple the problem *per user* and take the elements of S to be the pairs (ℓ, m) and repeat the steps of Lemma 3 and Theorem 4 to show submodularity and monotonicity for $f_{ki}^*(S)$. Since the objective $F^*(X)$ of Problem 2 is a positive weighted sum of $f_{ki}^*(S)$, it is monotone submodular as well.

Appendix B

Proofs of Chapter 4

B.1 Useful properties

This section groups a number of facts used in the proofs of Section B.2.

Fact 2. For two vectors $u, v \in \mathbb{R}^d$ and $\delta > 0$ it holds

$$-2\langle \mathbf{u}, \mathbf{v} \rangle \leq \frac{1}{\delta}\|\mathbf{u}\|_2^2 + \delta\|\mathbf{v}\|_2^2. \tag{B.1}$$

This also holds for matrices $u, v \in \mathbb{R}^{m\times n}$ and the Frobenius norm. It can be shown by manipulating the term $\left\|\frac{1}{\delta}\mathbf{u} + \delta\mathbf{v}\right\|_2^2 \geq 0$.

Fact 3. For two matrices $A, B \in \mathbb{R}^{m\times n}$ it holds that

$$\|A + B\|_F^2 \leq (1 + \alpha^{-1})\|A\|_F^2 + (1 + \alpha)\|B\|_F^2. \tag{B.2}$$

This can be shown using (B.1).

Fact 4. For a matrix $A \in \mathbb{R}^{m\times n}$ it can be shown that

$$\left\|A\left(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T\right)\right\|_F^2 \leq \|A\|_F^2. \tag{B.3}$$

Fact 5. Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix satisfying $\mathbf{1}^T M = \mathbf{1}$ and having eigenvalues $1 = \lambda_1 > |\lambda_2| \geq \ldots \geq |\lambda_n|$. Then, for a vector $\mathbf{x} \in \mathbb{R}^n$ with the average of its entries denoted $x_{\text{avg}} = \mathbf{1}^T\mathbf{x}$, it holds

$$(\mathbf{x} - x_{\text{avg}}\mathbf{1})^T M(\mathbf{x} - x_{\text{avg}}\mathbf{1}) \leq |\lambda_2|\,\|\mathbf{x} - x_{\text{avg}}\mathbf{1}\|_2^2. \tag{B.4}$$

This is a consequence of the spectral theorem and the fact that $(\mathbf{x} - x_{\text{avg}}\mathbf{1}) \perp \text{span}\{\mathbf{1}\}$.

B.2 Proofs of Lemmas

Proof of Lemma 17. It holds that

$$\begin{aligned}\mathbb{E}\left\|\overline{\mathbf{x}}^{t+1}-\mathbf{z}^*\right\|_2^2 &= \mathbb{E}\left\|\overline{\mathbf{z}}^t-\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t,\xi_i^t)-\mathbf{z}^*\right\|_2^2\\ &= \mathbb{E}\left\|\overline{\mathbf{z}}^t-\mathbf{z}^*-\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t)\right\|_2^2+\eta_t^2\,\mathbb{E}\left\|\frac{1}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t)-\frac{1}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t,\xi_i^{j_t})\right\|_2^2\\ &\quad+2\,\mathbb{E}\left\langle\overline{\mathbf{z}}^t-\mathbf{z}^*-\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t),\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t)-\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t,\xi_i^{j_t})\right\rangle \qquad \text{(B.5)}\end{aligned}$$

where in the first equality we used that

$$\overline{\mathbf{x}}^{t+1}=\frac{1}{n}\sum_{i=1}^n\mathbf{x}_i^{t+1}=\frac{1}{n}\sum_{i=1}^n\sum_{j=1}^n W_{ij}\mathbf{x}_i^{t+\frac{1}{2}}=\frac{1}{n}\sum_{i=1}^n\mathbf{x}_i^{t+\frac{1}{2}}\sum_{j=1}^n W_{ij}=\overline{\mathbf{x}}^{t+\frac{1}{2}}=\overline{\mathbf{z}}^t-\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t,\xi_i^t)$$

and in the second equality we added and subtracted $\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{x}_i^t)$ inside the norm.

Note that the last term in (B.5) is zero, since $\mathbb{E}_{\xi_i^t}[\nabla F_i(\mathbf{z}_i^t,\xi_i^t)]=\nabla F_i(\mathbf{z}_i^t)$, and the second term is bounded by $\eta_t^2\frac{\overline{\sigma}^2}{n}$ (eq. (4.6)). The first term in (B.5) can be written as (we will apply the expectation directly to the end result)

$$\begin{aligned}\left\|\overline{\mathbf{z}}^t-\mathbf{z}^*-\frac{\eta_t}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t)\right\|_2^2 &= \left\|\overline{\mathbf{z}}^t-\mathbf{z}^*\right\|_2^2+\eta_t^2\underbrace{\left\|\frac{1}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t)\right\|_2^2}_{(A)}\\ &\quad\underbrace{-2\eta_t\left\langle\overline{\mathbf{z}}^t-\mathbf{z}^*,\frac{1}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t)\right\rangle}_{(B)}. \qquad \text{(B.6)}\end{aligned}$$

We can bound term (A) with

$$\begin{aligned}\left\|\frac{1}{n}\sum_{i=1}^n\nabla F_i(\mathbf{z}_i^t)\right\|_2^2 &\le \frac{1}{n}\sum_{i=1}^n\left\|\nabla F_i(\mathbf{z}_i^t)\right\|_2^2\\ &= \frac{1}{n}\sum_{i=1}^n\left\|\nabla F_i(\mathbf{z}_i^t)-\nabla F_i(\mathbf{z}_i^*)\right\|_2^2\\ &\overset{(4.5)}{\le} \frac{1}{n}\sum_{i=1}^n 2L(F_i(\mathbf{z}_i^t)-F_i(\mathbf{z}_i^*)). \qquad \text{(B.7)}\end{aligned}$$

On the other hand, term (B) can be bounded as

$$-2\eta_t\left\langle \overline{\mathbf{z}}^t - \mathbf{z}^*, \frac{1}{n}\sum_{i=1}^n \nabla F_i(\mathbf{z}_i^t)\right\rangle = \frac{-2\eta_t}{n}\sum_{i=1}^n \left\langle \overline{\mathbf{z}}^t - \mathbf{z}_i^t + \mathbf{z}_i^t - \mathbf{z}^*, \nabla F_i(\mathbf{z}_i^t)\right\rangle$$

$$= \frac{-2\eta_t}{n}\sum_{i=1}^n \left\langle \overline{\mathbf{z}}^t - \mathbf{z}_i^t, \nabla F_i(\mathbf{z}_i^t)\right\rangle - \frac{2\eta_t}{n}\sum_{i=1}^n \left\langle \mathbf{z}_i^t - \mathbf{z}^*, \nabla F_i(\mathbf{z}_i^t)\right\rangle$$

$$\overset{(B.1),(4.2)}{\leq} \frac{\eta_t}{n}\sum_{i=1}^n \left(\frac{1}{\eta_t}\left\|\overline{\mathbf{z}}^t - \mathbf{z}_i^t\right\|_2^2 + \eta_t\left\|\nabla F_i(\mathbf{z}_i^t)\right\|_2^2\right)$$

$$- \frac{2\eta_t}{n}\sum_{i=1}^n \left((F_i(\mathbf{z}_i^t) - F_i(\mathbf{z}^*)) + \frac{\mu}{2}\left\|\mathbf{z}_i^t - \mathbf{z}^*\right\|_2^2\right).$$

where we applied (B.1) with the choice $\delta = \eta_t$.

Replacing the bounds on (A) and (B) in (B.5) gives

$$\left\|\overline{\mathbf{z}}^t - \mathbf{z}^* - \frac{\eta_t}{n}\sum_{i=1}^n \nabla F_i(\mathbf{z}_i^t)\right\|_2^2 \leq \left\|\overline{\mathbf{z}}^t - \mathbf{z}^*\right\|_2^2 + \frac{\eta_t^2}{n} 2L\sum_{i=1}^n (F_i(\mathbf{z}_i^t) - F_i(\mathbf{z}_i^*))$$

$$+ \frac{\eta_t}{n}\sum_{i=1}^n \left(\frac{1}{\eta_t}\left\|\overline{\mathbf{z}}^t - \mathbf{z}_i^t\right\|_2^2 + \eta_t\left\|\nabla F_i(\mathbf{z}_i^t)\right\|_2^2\right) - \frac{2\eta_t}{n}\sum_{i=1}^n \left((F_i(\mathbf{z}_i^t) - F_i(\mathbf{z}^*)) + \frac{\mu}{2}\left\|\mathbf{z}_i^t - \mathbf{z}^*\right\|_2^2\right)$$

$$\overset{(B.7)}{\leq} (1 - \mu\eta_t)\left\|\overline{\mathbf{z}}^t - \mathbf{z}^*\right\|_2^2 + \frac{1}{n}\sum_{i=1}^n \left\|\overline{\mathbf{z}}^t - \mathbf{z}_i^t\right\|_2^2$$

$$\underbrace{+ \eta_t^2 \frac{4L}{n}\sum_{i=1}^n (F_i(\mathbf{z}_i^t) - F_i(\mathbf{z}_i^*)) - \frac{2\eta_t}{n}\sum_{i=1}^n (F_i(\mathbf{z}_i^t) - F_i(\mathbf{z}^*))}_{(C)}$$

where we used

$$-\frac{\eta_t\mu}{n}\sum_{i=1}^n \left\|\mathbf{z}_i^t - \mathbf{z}^*\right\|_2^2 \leq -\eta_t\mu\left\|\frac{1}{n}\sum_{i=1}^n (\mathbf{z}_i^t - \mathbf{z}^*)\right\|_2^2 = -\eta_t\mu\left\|\overline{\mathbf{z}}^t - \mathbf{z}^*\right\|_2^2.$$

Term (C) was bounded in the proof of Lemma 1 of [50], where for $\eta_t \leq \frac{1}{4L}$ and $\Gamma = \frac{1}{n}\sum_{i=1}^n (F_i(\mathbf{z}^*) - F_i(\mathbf{z}_i^*))$ they obtained

$$(C) \leq \frac{1}{n}\sum_{i=1}^n \left\|\mathbf{z}_i^t - \overline{\mathbf{z}}^t\right\|_2^2 + 6L\eta_t^2\Gamma.$$

Replacing these bounds in (B.5) gives the lemma. □

Proof of Lemma 18. We define

$$Z^t = \left[\mathbf{z}_1^t \cdots \mathbf{z}_n^t\right] \in \mathbb{R}^{d\times n}$$

$$\overline{Z}^t = Z^t \frac{1}{n}\mathbf{1}\mathbf{1}^T = \left[\overline{\mathbf{z}}^t \cdots \overline{\mathbf{z}}^t\right] \in \mathbb{R}^{d\times n}$$
$$\partial F(Z^t, \xi^t) = \left[\nabla F_1(\mathbf{z}_1^t, \xi_1^t) \cdots \nabla F_n(\mathbf{z}_n^t, \xi_n^t)\right] \in \mathbb{R}^{d\times n}$$

and $X^t, \overline{X}^t$ analogously. Note that with these definitions, $\sum_{i=1}^n \|\mathbf{z}_i^t - \overline{\mathbf{z}}^t\|_2^2 = \left\|Z^t - \overline{Z}^t\right\|_F^2$.

We denote $t_b \in \mathcal{H}$ the last time the central server broadcasted the sample average to all nodes so that $\mathbf{z}_i^{t_b} = \mathbf{z}^{t_b}\ \forall i \in [n]$, and define $h := t - t_b \leq (H-1)$. Note that if $t = t_b$ then $\sum_{i=1}^n \|\mathbf{z}_i^t - \overline{\mathbf{z}}\|_2^2 = 0$. Therefore, below we assume $h \geq 1$. We have that

$$\begin{aligned}
&\mathbb{E}\left\|Z^t - \overline{Z}^t\right\|_F^2 \overset{t\notin\mathcal{H}}{=} \mathbb{E}\left\|X^t - \overline{X}^t\right\|_F^2 = \mathbb{E}\Big\|\underbrace{(X^{t-\frac{1}{2}} - \overline{X}^{t-\frac{1}{2}})}_{Y} W^{t-1}\Big\|_F^2 \\
&= \mathbb{E}\sum_{i=1}^d \left\|Y_{[i,:]}W^{t-1}\right\|_2^2 = \mathbb{E}\sum_{i=1}^d Y_{[i,:]}\mathbb{E}_W\left[WW^T\right][Y_{[i,:]}]^T \\
&\overset{(B.4)}{\leq} \mathbb{E}\sum_{i=1}^d Y_{[i,:]} \underbrace{\left|\lambda_2\left(\mathbb{E}_W\left[WW^T\right]\right)\right|}_{|\widehat{\lambda}_2|} [Y_{[i,:]}]^T = |\widehat{\lambda}_2|\ \mathbb{E}\left\|X^{t-\frac{1}{2}} - \overline{X}^{t-\frac{1}{2}}\right\|_F^2 \\
&= |\widehat{\lambda}_2|\ \mathbb{E}\left\|Z^{t-1} - \overline{Z}^{t-1} - \eta_{t-1}\partial F(Z^{t-1}, \xi^{t-1})(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)\right\|_F^2 \\
&\overset{(B.2)}{\leq} |\widehat{\lambda}_2|\left(1+\frac{1}{\alpha}\right)\mathbb{E}\left\|Z^{t-1} - \overline{Z}^{t-1}\right\|_F^2 + |\widehat{\lambda}_2|(1+\alpha)\eta_{t-1}^2\mathbb{E}\left\|\partial F(Z^{t-1}, \xi^{t-1})(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T)\right\|_F^2 \\
&\overset{(B.3)}{\leq} |\widehat{\lambda}_2|\left(1+\frac{1}{\alpha}\right)\mathbb{E}\left\|Z^{t-1} - \overline{Z}^{t-1}\right\|_F^2 + |\widehat{\lambda}_2|\ (1+\alpha)\eta_{t-1}^2\mathbb{E}\left\|\partial F(Z^{t-1}, \xi^{t-1})\right\|_F^2 \\
&= |\widehat{\lambda}_2|\left(1+\frac{1}{\alpha}\right)\mathbb{E}\left\|Z^{t-1} - \overline{Z}^{t-1}\right\|_F^2 + |\widehat{\lambda}_2|(1+\alpha)\eta_{t-1}^2 nG^2,
\end{aligned}$$

where in the second line we used that $\mathbf{1}^T W^t = \mathbf{1}^T$, and in the fourth line, that the matrices W^t are identically distributed independently of the time t. We also used $Y_{[i,:]}$ to indicate the i-th row of matrix Y.

We can now apply the inequality recursively to get

$$\begin{aligned}
&\mathbb{E}\left\|Z^t - \overline{Z}^t\right\|_F^2 \leq \left(1+\frac{1}{\alpha}\right)|\widehat{\lambda}_2|\ \mathbb{E}\left\|Z^{t-1} - \overline{Z}^{t-1}\right\|_F^2 + (1+\alpha)|\widehat{\lambda}_2|\ \eta_{t-1}^2 nG^2 \\
&\leq \left(1+\frac{1}{\alpha}\right)|\widehat{\lambda}_2|\ \left[\left(1+\frac{1}{\alpha}\right)|\widehat{\lambda}_2|\ \mathbb{E}\left\|Z^{t-2} - \overline{Z}^{t-2}\right\|_F^2 + (1+\alpha)|\widehat{\lambda}_2|\ \eta_{t-2}^2 nG^2\right] \\
&\qquad + (1+\alpha)|\widehat{\lambda}_2|\ \eta_{t-1}^2 nG^2 \\
&= \left[\left(1+\frac{1}{\alpha}\right)|\widehat{\lambda}_2|\ \right]^2 \mathbb{E}\left\|Z^{t-2} - \overline{Z}^{t-2}\right\|_F^2 + (1+\alpha)|\widehat{\lambda}_2|\ nG^2 \sum_{i=1}^2 \left[\left(1+\frac{1}{\alpha}\right)|\widehat{\lambda}_2|\ \right]^{i-1} \eta_{t-i}^2 \\
&\leq \left[\left(1+\frac{1}{\alpha}\right)|\widehat{\lambda}_2|\ \right]^h \mathbb{E}\left\|Z^{t_b} - \overline{Z}^{t_b}\right\|_F^2 + (1+\alpha)|\widehat{\lambda}_2|\ nG^2 \sum_{i=1}^h \left[\left(1+\frac{1}{\alpha}\right)|\widehat{\lambda}_2|\ \right]^{i-1} \eta_{t-i}^2.
\end{aligned}$$

We note that the first term is zero, since at broadcasting time $Z^{t_b} = \overline{Z}^{t_b}$. We now set

$\alpha = \frac{|\widehat{\lambda}_2|}{1-|\widehat{\lambda}_2|}$ so that the expression between square brackets takes value 1. Therefore,

$$\mathbb{E}\left\|Z^t - \overline{Z}^t\right\|_F^2 \leq \alpha n G^2 H \eta_{t_b}^2 \leq \alpha n G^2 H 4 \eta_t^2$$

where we have used that for the choice of α given above it holds $(1+\alpha)|\widehat{\lambda}_2| = \alpha$, and that the stepsizes η_t are monotonically decreasing and satisfy $\eta_t \leq 2\eta_{t+H}$. □

Note that in the result above we could have used $(H-1)$ instead of H. However, since this bound is used again in the proof of Lemma 19 for $h = H$, we loosen it slightly here to be able to apply it directly in the next proof.

Proof of Lemma 19. For $t \in \mathcal{H}$ we have that

$$\mathbb{E}\left\|\overline{\mathbf{z}}^t - \overline{\mathbf{x}}^t\right\|_2^2 = \mathbb{E}\left\|\frac{1}{K}\sum_{\ell=1}^{K}\mathbf{x}_{j_\ell}^t - \overline{\mathbf{x}}^t\right\|_2^2 = \frac{1}{K^2}\sum_{\ell=1}^{K}\mathbb{E}\left[\mathbb{E}_{\mathcal{S}_t}\left\|\mathbf{x}_{j_\ell}^t - \overline{\mathbf{x}}^t\right\|_2^2\right] = \frac{1}{Kn}\sum_{i=1}^{n}\mathbb{E}\left\|\mathbf{x}_i^t - \overline{\mathbf{x}}^t\right\|_2^2$$

where we used that for independent X_i, $\mathrm{Var}(\sum_i X_i) = \sum_i \mathrm{Var}(X_i)$, and that for a random variable $X \in \Omega$ with a discrete probability density function $X \sim f_X$ and a function $g : \Omega \to \mathbb{R}$ it holds $\mathbb{E}[g(X)] = \sum_x g(x) f_X(x)$.

Since $\sum_{i=1}^n \|\mathbf{x}_i^t - \overline{\mathbf{x}}^t\|_2^2 = \left\|X^t - \overline{X}^t\right\|_F^2$ we can repeat the procedure done in Lemma 18 to bound this term. □

Bibliography

[1] M. Roser, H. Ritchie, and E. Ortiz-Ospina, "Internet," *Our World in Data*, 2015. https://ourworldindata.org/internet.

[2] Cisco Systems, "Cisco annual internet report (2018–2023) white paper." https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. Retrieved on March 2023.

[3] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[4] J. O. du Terrail, A. Leopold, C. Joly, C. Béguier, M. Andreux, C. Maussion, B. Schmauch, E. W. Tramel, E. Bendjebbar, M. Zaslavskiy, *et al.*, "Federated learning for predicting histological response to neoadjuvant chemotherapy in triple-negative breast cancer," *Nature medicine*, vol. 29, no. 1, pp. 135–146, 2023.

[5] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of biomedical informatics*, vol. 50, pp. 234–243, 2014.

[6] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," *Journal of biomedical informatics*, vol. 99, p. 103291, 2019.

[7] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, M. D. Mueck, and S. Srikanteswara, "Energy demand prediction with federated learning for electric vehicle networks," in *2019 IEEE global communications conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[8] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, "Ffd: A federated learning based method for credit card fraud detection," in *Big Data–BigData 2019: 8th International Congress, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 8*, pp. 18–32, Springer, 2019.

[9] X. Han, H. Yu, and H. Gu, "Visual inspection with federated learning," in *Image Analysis and Recognition: 16th International Conference, ICIAR 2019, Waterloo, ON, Canada, August 27–29, 2019, Proceedings, Part II 16*, pp. 52–64, Springer, 2019.

[10] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless video content delivery through distributed caching helpers," in *2012 Proceedings IEEE INFOCOM*, pp. 1107–1115, March 2012.

[11] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5g networks with mobile edge computing," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 80–87, 2018.

[12] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at" the edge" of information-centric networks," in *Proceedings of the 1st ACM conference on information-centric networking*, pp. 47–56, 2014.

[13] D. T. Hoang, D. Niyato, D. N. Nguyen, E. Dutkiewicz, P. Wang, and Z. Han, "A dynamic edge caching framework for mobile 5g networks," *IEEE Wireless Communications*, vol. 25, no. 5, pp. 95–103, 2018.

[14] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networking," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–8, IEEE, 2014.

[15] "Netflix open connect." https://openconnect.netflix.com/en/.

[16] "Google global cache." https://peering.google.com/#/.

[17] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems (TMIS)*, vol. 6, no. 4, p. 13, 2016.

[18] R. Zhou, S. Khemmarat, and L. Gao, "The impact of youtube recommendation system on video views," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 404–410, ACM, 2010.

[19] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1300–1313, 2018.

[20] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[21] P. Wan and M. D. Lemmon, "Event-triggered distributed optimization in sensor networks," in *2009 International Conference on Information Processing in Sensor Networks*, pp. 49–60, IEEE, 2009.

[22] M. Alrowaily and Z. Lu, "Secure edge computing in IoT systems: review and case studies," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 440–444, IEEE, 2018.

[23] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, N. A. Aziz, *et al.*, "Swarm learning for decentralized and confidential clinical machine learning," *Nature*, vol. 594, no. 7862, pp. 265–270, 2021.

[24] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[25] E. Wei and A. Ozdaglar, "On the $O(1/k)$ convergence of asynchronous distributed alternating direction method of multipliers," in *2013 IEEE Global Conference on Signal and Information Processing*, pp. 551–554, IEEE, 2013.

[26] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic control*, vol. 57, no. 3, pp. 592–606, 2012.

[27] D. Jakovetić, J. Xavier, and J. M. Moura, "Fast distributed gradient methods," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1131–1146, 2014.

[28] W. Shi, Q. Ling, G. Wu, and W. Yin, "Extra: An exact first-order algorithm for decentralized consensus optimization," *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 944–966, 2015.

[29] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *International Conference on Machine Learning*, pp. 3478–3487, PMLR, 2019.

[30] G. Nadiradze, A. Sabour, P. Davies, S. Li, and D. Alistarh, "Asynchronous decentralized sgd with quantized and local updates," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6829–6842, 2021.

[31] S. Pu, W. Shi, J. Xu, and A. Nedić, "Push–pull gradient methods for distributed optimization in networks," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 1–16, 2020.

[32] V. Khatana, G. Saraswat, S. Patel, and M. V. Salapaka, "Gradient-consensus method for distributed optimization in directed multi-agent networks," in *2020 American Control Conference (ACC)*, pp. 4689–4694, IEEE, 2020.

[33] A. Nedic, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM Journal on Optimization*, vol. 27, no. 4, pp. 2597–2633, 2017.

[34] L. He, A. Bian, and M. Jaggi, "Cola: Decentralized linear learning," *Advances In Neural Information Processing Systems 31 (Nips 2018)*, vol. 31, no. CONF, 2018.

[35] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié, "Optimal algorithms for smooth and strongly convex distributed optimization in networks," in *international conference on machine learning*, pp. 3027–3036, PMLR, 2017.

[36] D. Jakovetic, D. Bajovic, A. K. Sahu, and S. Kar, "Convergence rates for distributed stochastic optimization over random networks," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 4238–4245, IEEE, 2018.

[37] B. Li, S. Cen, Y. Chen, and Y. Chi, "Communication-efficient distributed optimization in networks with gradient tracking and variance reduction," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 7331–7381, 2020.

[38] C. A. Uribe, S. Lee, A. Gasnikov, and A. Nedić, "A dual approach for optimal algorithms in distributed optimization over networks," in *2020 Information Theory and Applications Workshop (ITA)*, pp. 1–37, IEEE, 2020.

[39] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *52nd IEEE Conference on Decision and Control (CDC)*, pp. 3671–3676, IEEE, 2013.

[40] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "Convergence of asynchronous distributed gradient methods over stochastic networks," *IEEE Transactions on Automatic Control*, vol. 63, no. 2, pp. 434–448, 2017.

[41] K. Srivastava and A. Nedic, "Distributed asynchronous constrained stochastic optimization," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 772–790, 2011.

[42] S. S. Ram, A. Nedić, and V. V. Veeravalli, "Asynchronous gossip algorithms for stochastic optimization," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 3581–3586, IEEE, 2009.

[43] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.

[44] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.

[45] J. Nutini, M. Schmidt, I. Laradji, M. Friedlander, and H. Koepke, "Coordinate descent converges faster with the Gauss-Southwell rule than random selection," in *International Conference on Machine Learning*, pp. 1632–1641, PMLR, 2015.

[46] J. Nutini, I. Laradji, and M. Schmidt, "Let's make block coordinate descent go fast: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence," *arXiv preprint arXiv:1712.08859*, 2017.

[47] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[48] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[49] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*, pp. 5132–5143, PMLR, 2020.

[50] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on non-iid data," in *International Conference on Learning Representations*, 2020.

[51] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of local-update sgd algorithms," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 9709–9758, 2021.

[52] S. U. Stich and S. P. Karimireddy, "The error-feedback framework: Better rates for SGD with delayed gradients and compressed updates," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 9613–9648, 2020.

[53] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, "Local sgd with periodic averaging: Tighter analysis and adaptive synchronization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[54] S. U. Stich, "Local SGD converges fast and communicates little," in *ICLR 2019-International Conference on Learning Representations*, 2019.

[55] H. Hellaoui, O. Bekkouche, M. Bagaa, and T. Taleb, "Aerial control system for spectrum efficiency in uav-to-cellular communications," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 108–113, 2018.

[56] A. Asadi, Q. Wang, and V. Mancuso, "A survey on device-to-device communication in cellular networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1801–1819, 2014.

[57] M. Yemini, R. Saha, E. Ozfatura, D. Gündüz, and A. J. Goldsmith, "Semi-decentralized federated learning with collaborative relaying," in *2022 IEEE International Symposium on Information Theory (ISIT)*, pp. 1471–1476, IEEE, 2022.

[58] F. P.-C. Lin, S. Hosseinalipour, S. S. Azam, C. G. Brinton, and N. Michelusi, "Semi-decentralized federated learning with cooperative D2D local model aggregations," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3851–3869, 2021.

[59] L. Chou, Z. Liu, Z. Wang, and A. Shrivastava, "Efficient and less centralized federated learning," in *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*, pp. 772–787, Springer, 2021.

[60] S. Hosseinalipour, S. S. Azam, C. G. Brinton, N. Michelusi, V. Aggarwal, D. J. Love, and H. Dai, "Multi-stage hybrid federated learning over large-scale d2d-enabled fog networks," *IEEE/ACM Transactions on Networking*, vol. 30, no. 4, pp. 1569–1584, 2022.

[61] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich, "A unified theory of decentralized SGD with changing topology and local updates," in *International Conference on Machine Learning*, pp. 5381–5393, PMLR, 2020.

[62] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, "The role of caching in future communication systems and networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1111–1125, 2018.

[63] P. Lyman and H. R. Varian, "How Much Information 2003." http://groups.ischool.berkeley.edu/archive/how-much-info-2003/. Retrieved on February 2020.

[64] T. V. Doan, L. Pajevic, V. Bajpai, and J. Ott, "Tracing the path to youtube: A quantification of path lengths and latencies toward content caches," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 80–86, 2018.

[65] T. V. Doan, L. Pajevic, V. Bajpai, and J. Ott, "Tracing the path to youtube: A quantification of path lengths and latencies toward content caches," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 80–86, 2019.

[66] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[67] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen, "Cache-centric video recommendation: an approach to improve the efficiency of youtube caches," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 4, p. 48, 2015.

[68] T. Giannakas, P. Sermpezis, and T. Spyropoulos, "Show me the cache: Optimizing cache-friendly recommendations for sequential content access," in *19th IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks", WoWMoM 2018, Chania, Greece, June 12-15, 2018*, pp. 14–22, 2018.

[69] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Caching-aware recommendations: Nudging user preferences towards better caching performance," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9, IEEE, 2017.

[70] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Jointly optimizing content caching and recommendations in small cell networks," *IEEE Trans. Mob. Comput.*, vol. 18, no. 1, pp. 125–138, 2019.

[71] Z. Lin and W. Chen, "Joint pushing and recommendation for susceptible users with time-varying connectivity," in *GLOBECOM*, pp. 1–6, 2018.

[72] L. Song and C. Fragouli, "Making recommendations bandwidth aware," *IEEE Trans. Information Theory*, vol. 64, no. 11, pp. 7031–7050, 2018.

[73] H. Nam, K.-H. Kim, and H. Schulzrinne, "Qoe matters more than qos: Why people stop watching cat videos," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, 2016.

[74] T. Spyropoulos and P. Sermpezis, "Soft cache hits and the impact of alternative content recommendations on mobile edge caching," in *Proc. ACM Workshop on Challenged Networks (CHANTS)*, pp. 51–56, 2016.

[75] R. Zhou, S. Khemmarat, and L. Gao, "The impact of youtube recommendation system on video views," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 404–410, ACM, 2010.

[76] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3665–3677, 2014.

[77] A. Krause and D. Golovin, "Submodular function maximization," *Tractability: Practical Approaches to Hard Problems*, vol. 3, no. 19, p. 8, 2012.

[78] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet.," *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[79] N. Golrezaei, K. Shanmugam, A. Dimakis, A. Molisch, and G. Caire, "Femtocaching: Wireless video content delivery through distributed caching helpers," in *IEEE INFOCOM*, pp. 1107–1115, March 2012.

[80] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[81] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions—ii," in *Polyhedral combinatorics*, pp. 73–87, Springer, 1978.

[82] "Dataset for statistics and social network of youtube videos." http://netsg.cs.sfu.ca/youtubedata/, 2008.

[83] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52, ACM, 2015.

[84] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[85] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[86] A. Kehagias, "Community detection toolbox." https://www.mathworks.com/matlabcentral/fileexchange/45867-community-detection-toolbox), MATLAB Central File Exchange. Retrieved on November 2018.

[87] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[88] "Million song dataset." http://millionsongdataset.com/lastfm/.

[89] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001.

[90] A. Nedic and A. Ozdaglar, "On the rate of convergence of distributed subgradient methods for multi-agent optimization," in *2007 46th IEEE Conference on Decision and Control*, pp. 4711–4716, IEEE, 2007.

[91] G. Neglia, C. Xu, D. Towsley, and G. Calbi, "Decentralized gradient methods: does topology matter?," in *International Conference on Artificial Intelligence and Statistics*, pp. 2348–2358, PMLR, 2020.

[92] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *International Conference on Machine Learning*, pp. 3043–3052, PMLR, 2018.

[93] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1835–1854, 2016.

[94] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Transactions on Automatic Control*, vol. 31, no. 9, pp. 803–812, 1986.

[95] Z. Peng, Y. Xu, M. Yan, and W. Yin, "Arock: an algorithmic framework for asynchronous parallel coordinate updates," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. A2851–A2879, 2016.

[96] L. Xiao, A. W. Yu, Q. Lin, and W. Chen, "DSCOVR: Randomized primal-dual block coordinate algorithms for asynchronous distributed optimization," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1634–1691, 2019.

[97] A. Verma, M. M. Vasconcelos, U. Mitra, and B. Touri, "Maximal dissent: a state-dependent way to agree in distributed convex optimization," *IEEE Transactions on Control of Network Systems*, 2023.

[98] Y. You, X. Lian, J. Liu, H.-F. Yu, I. S. Dhillon, J. Demmel, and C.-J. Hsieh, "Asynchronous parallel greedy coordinate descent," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[99] H. Hendrikx, F. Bach, and L. Massoulié, "Accelerated decentralized optimization with local updates for smooth and strongly convex objectives," in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 897–906, PMLR, 2019.

[100] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[101] R. T. Rockafellar and R. J.-B. Wets, *Variational analysis*, vol. 317. Springer Science & Business Media, 2009.

[102] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M.

Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[103] Z. Qu, K. Lin, Z. Li, and J. Zhou, "Federated learning's blessing: Fedavg has linear speedup," in *ICLR 2021-Workshop on Distributed and Private Machine Learning (DPML)*, 2021.

[104] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in neural information processing systems*, vol. 24, 2011.

[105] J. Liu, S. Wright, C. Ré, V. Bittorf, and S. Sridhar, "An asynchronous parallel stochastic coordinate descent algorithm," in *International Conference on Machine Learning*, pp. 469–477, PMLR, 2014.

[106] V. Smith, S. Forte, M. Chenxin, M. Takác, M. I. Jordan, and M. Jaggi, "CoCoA: A general framework for communication-efficient distributed optimization," *Journal of Machine Learning Research*, vol. 18, p. 230, 2018.

[107] J. Zhang, C. De Sa, I. Mitliagkas, and C. Ré, "Parallel SGD: When does averaging help?," *arXiv preprint arXiv:1606.07365*, 2016.

[108] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[109] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[110] T. Vogels, H. Hendrikx, and M. Jaggi, "Beyond spectral gap: The role of the topology in decentralized learning," *arXiv preprint arXiv:2206.03093*, 2022.

[111] M. Barthélemy, "Spatial networks," *Physics reports*, vol. 499, no. 1-3, pp. 1–101, 2011.

[112] M. E. Newman, "The structure and function of complex networks," *SIAM review*, vol. 45, no. 2, pp. 167–256, 2003.

[113] F. Gabry, V. Bioglio, and I. Land, "On energy-efficient edge caching in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3288–3298, 2016.

[114] G. Ahani and D. Yuan, "Optimal content caching and recommendation with age of information," *IEEE Transactions on Mobile Computing*, 2022.

[115] T. Vogels, S. P. Karimireddy, and M. Jaggi, "PowerSGD: Practical low-rank gradient compression for distributed optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.