An Accelerated Algorithm for Delayed Distributed Convex Optimization

Ioannis M. Bakagiannis

Master of Engineering

Department of Electrical and Computer Engineering

McGill University

Montreal,Quebec

2015-08-04

A thesis submitted to McGill University
in partial fulfilment of the requirements of the degree of
Master of Engineering

DEDICATION

To my family who has always been there for me.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincerest gratitude to my advisor Professor Michael Rabbat. I thank him for offering me the chance to be part of his research group, for his honest advice in difficult times, and for providing me with full support during my studies at McGill thus far. I would also like to thank my family in Greece and in Canada for believing in my decisions and providing all the tools to stand up on my own. Also I would like to thank my friends Argyris, Kostas, Antreas, Olympia, Prokopis, Konstantinos, Jim and many more from all over the world that stood by me in all stages of my life and for supporting me every day.

ABSTRACT

In many large-scale optimization problems arising in the context of machine learning the decision variable is of high-dimension and the objective function decomposes into a sum over a large number of terms (one for each instance in the training data set).In this setting, second-order optimization methods such as Newton or quasi-Newton methods, are not tractable due to the complexity of evaluating and inverting the Hessian, or an approximation thereof. Also, the vast amount of data available is spread around multiple servers making a centralized optimization solution sub optimal or impossible. Therefore we concentrate on first order methods that are scalable in a decentralized setting. In this thesis we provide a framework for distributed delayed convex optimization methods for networks in a master-server setting. Our goal is to optimize a global objective function which is the sum of the local objective functions of the agents in the network.We review Nesterov's accelerated algorithm for centralized optimization since it is the optimal algorithm for the class of convex, and strongly convex functions and to modify it accordingly for decentralized optimization in the master-server setting. It is natural that in an asynchronous setting the current value of the server node is a past value of the master node communicated some time steps ago, and thus gives rise to delays in the analysis. We have proven that a delayed accelerated method maintains the optimality of the algorithm with a convergence rate of $O\left(\frac{1}{t^2}\right)$. We have also performed simulations and we have verified that

the accelerated algorithm performs better that the alternative algorithms for decentralized optimization in a master server setting.

ABRÉGÉ

Dans de nombreux problèmes d'optimisation à grande échelle qui se posent dans le cadre de la machine l'apprentissage , la variable de décision est de haute dimension et la fonction objectif se décompose en une somme sur un grand nombre de termes (une pour chaque instance dans l'ensemble de données d'apprentissage). Dans ce cadre , les méthodes d'optimisation de second ordre tels en tant que méthode de Newton ou quasi- Newton , ne sont pas traitables par la complexité de et l'évaluation de l'inversion de la Hessian ou une approximation de celui-ci. Par conséquent , nous nous concentrons sur les méthodes du premier ordre qui sont évolutives dans un cadre décentralisé. Dans cette thèse, nous fournissons un système cadre pour des méthodes d'optimisation distribués retardés convexes pour les réseaux multi-agents dans un cadre serveur-maître. Notre objectif est d' optimiser une fonction objectif globale qui est la somme des fonctions objectifs locales des agents dans le réseau. Nous introduisons l'idée d' accélérer l'algorithme de Nesterov pour l'optimisation centralisée dans le cadre serveur maître décentralisée puisqu'il est l'algorithme optimal pour le classe de fonctions qui nous intéressent. Il est naturel que dans un cadre asynchrone, la valeur actuelle du noeud de serveur est une valeur passée du noeud maître communiquée depuis quelques mesures de temps et ainsi donner lieu à des retards dans l'analyse. Nous ont prouvé que d'une méthode accélérée retardé maintient l'optimalité de l'algorithme avec des taux de convergence de $O\left(\frac{1}{t^2}\right)$. Nous avons également effectué des simulations et nous avons vérifié qu'un algorithme accéléré surpasse

d'autres algorithmes disponible pour optimisation décentralisé dans un cadre de serveur maître.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Motivation

In modern world optimization, problems arise in various levels in almost every application field, such as machine learning, finance and network systems. It is human nature to try to go "faster" or try to use the least amount of resources to achieve a goal. The question that needs to be answered every time is how to minimize our costs in the quickest possible way. Depending on the field of application the problem could range from an optimal control problem to an optimal route to come back from work.

The mathematical formulation of all optimization problems is summarized in:

$$\min_{x \in R^n} f(x), \tag{1.1}$$

where f is the objective function of the problem at hand, which takes on many forms, and we can cast various assumptions over it. Throughout the thesis we assume that the function is convex, or strongly convex, depending on the algorithm that we analyze each time. Many fundamental convex optimization problems, especially in machine learning, take the form of:

$$\min_{x \in R^n} \sum_{i=1}^{m} f_i(x) + \lambda R(x), \tag{1.2}$$

where the functions $f_1 ... f_m, R$ are convex and $\lambda \geq 0$ is a fixed parameter. Each f_i can be interpreted as a cost of using x on the i-th element of a data set and $R(x)$ is a regularization term that helps keep x "simple". In many machine learning tasks (supervised learning problems) there is a data set of the form $(w_i, y_i) \in R^n \times Y$. In classification problems we have $Y = \{-1, 1\}$ and the cost function can be turned into hinge loss that is expressed as:

$$\min_{x \in R^n} \sum_{i=1}^{m} \max(0, 1 - y_i x^T w_i) + \lambda \|x\|_2^2, \tag{1.3}$$

to solve the Support Vector Machine problem or the cost function becomes the logistic loss

$$\min_{x \in R^n} \sum_{i=1}^{m} \log(1 + \exp(-y_i x^T w_i)) + \lambda \|x\|_2^2, \tag{1.4}$$

and obtain the logistic regression problem. In both cases we used L2 regularization in order to avoid over-fitting. In regression $Y = R$ and one of the most common problems is the least squares problem:

$$\min_{x \in R^n} \|Wx - Y\|_2^2, \tag{1.5}$$

or the Least Absolute Shrinkage and Selection Operator problem with L1 regularization:

$$\min_{x \in R^n} \|Wx - Y\|_2^2 + \|x\|_1. \tag{1.6}$$

All these machine learning problems fall under the spectrum of convex optimization.

In addition to the broad spectrum of optimization problems in recent years, we have an increasing amount of data that is needed for the optimization procedure. The expansion of the Internet and global connectivity has helped greatly in that regard. Also, imagine a portfolio optimization with stocks from diverse markets where data is flowing in real time. Now we can obtain, store and use all the information due to the advancement in technology, in vast amounts. Therefore, along with the optimization problem arises the problem of effectively using all the information, which is an optimization problem in itself. Our goal is to solve the optimization problem while using the least amount of resources as fast as possible.

One way to approach the processing of vast amounts of information is to divide it among several processing units. Although centralized approaches provide simpler solutions and better guarantees, as we will see in the following sections, they are limited by the sole unit itself. This means that the whole optimization procedure will depend on the capabilities of the one processing unit, which even if it is powerful enough it is destined to operate in a linear fashion that cannot do more than one operation at a time. Since the amount of data is ever growing it is unavoidable to turn to solutions that make use of multiple processing units and deal with a fraction of the data with each one. Also the current semiconductor technology is reaching the limits of Moore's law, and consequently the speed of individual processing units is no longer getting faster. To perform more calculations in a fixed amount of time we thus must turn to distributed processing, and this is why the technology is driven to multi-core processors and computer clusters.

There are many algorithms for solving convex optimization problems such as: gradient descent, mirror descent, Newton, quasi-Newton and many more. One way to categorize them is by the amount of information they require from the function itself. The two most popular categories are the first and second order methods. The former requires the value of the function and the first derivative of the function at point and the latter additionally requires the second derivative of the function. In large scale problems where the information is abundant and in problems that the dimension of the parameters is large, the computation alone of the computation of the second derivative is computationally intensive, or not-tractable, and it is needed in each iteration. Thus for scalability reasons we will contain our research to first order methods that are simpler in each iteration.

We already have to choose how to set up our approach towards the optimization problem: Centralized, which means that one processing unit holds all the information and performs the necessary optimization steps itself, or decentralized when you split the information into pieces and feed it to different processing units in order that each one performs an optimization step. Throughout this thesis we will examine both of these approaches and see their merits and their flaws.

For the decentralized setting, the information is distributed among multiple agents in a network and usually the network is quite large and the structure of it is given a priori. Each agent is given a convex function as their objective function and they cooperate with each other to solve the convex optimization problem through local information and communication over the network. Mathematically

this can be expressed as

$$\min_{x \in R^n} \frac{1}{m} \sum_{i=1}^{m} f_i(x) \tag{1.7}$$

where m is the total number of agents in the network and f_i the convex objective of each agent. At every time step through the optimization procedure each agent i maintains information x_i and tries to obtain x^* which is the optimal for the overall objective.

This decentralized setting implies that each agent in the network maintains data necessary to compute $f_i(x)$ and compute $\nabla f_i(x)$. Even if a single server maintains the current iterate of the decision vector and orchestrates gradient evaluations; there will be an inherent delay in querying the machines. Moreover, when the network latency and work load on the machines change, so will the query delay. It is therefore important that techniques developed for this set-up can handle time-varying delays [6].

Furthermore we are concerned about the convergence rate of the algorithm that tries to solve the convex optimization problem. By convergence rate we mean the rate that the sequence x_k, where k is the k-th time step of the algorithm, converges to x^* which is the solution of:

$$f(x^*) = \min_{x \in R^n} f(x). \tag{1.8}$$

Generally the convergence rate tells us how fast the algorithm is at hand. Especially for the centralized case of algorithms for convex optimization there have been many attempts to make these algorithms converge faster. For this cause the

accelerated algorithms were introduced by Nesterov [12] that achieves optimal convergence rate for the class of convex functions and speeds up the procedure by an order of magnitude compared to the other first order algorithms. In the decentralized field such optimality has not been achieved and more and more research is headed towards this purpose, such as ours.

In any distributed algorithm, it is necessary to coordinate to some extent the activities of the different processors. This coordination is often implemented by dividing the algorithm in "phases". During each phase, every processor executes a number of computations that depend on results of previous phases of other processors. But within a phase each processor doesn't interact with other processors as far as the given algorithm is concerned. All interaction takes place at the end of phases. Such algorithms are called synchronous and the "opposite" are the asynchronous algorithms where there are no phases and the coordination between processors is not strict [3]. Synchronous algorithms are more convenient to use since there is an order of execution and conflicts are resolved in a "tidy" manner. Unfortunately this convenience comes with a price as long as the time until convergence is concerned. It is obvious that for synchronous algorithms agents have to finish transmitting and receiving information in order to proceed to further computations. This creates a bottleneck and hurts synchronous algorithms scalability because this problem becomes bigger as the network becomes larger. Asynchronous algorithms are free from this curse and therefore are better suited for large scale problems despite the harder implementation due to the inherent complexity they have.

1.2 Contribution

Given the motivation for algorithms that can solve a convex optimization problem in a decentralized manner the focus of this thesis is twofold. First our goal is to provide an algorithm based on the accelerated algorithm of Nesterov that could handle delayed information. Asynchronous implementations of algorithms give birth to delays due to communication overhead or diversity in processing power. Also, as we will see in next chapter, Nesterov's accelerated algorithm is the optimal algorithm for the centralized setting and we wish to modify and implement it in a decentralized setting in order to obtain a faster convergence rate than the existing algorithms for decentralized optimization. This is a first step to observing how the accelerated algorithms behave when delayed information is forced upon them and what the are requirements for them to converge to the optimal solution. After defining the accelerated algorithm we proceed with a convergence proof which enables us to fine tune the parameters of the algorithm and prove an optimal convergence rate for the class of convex functions.

Secondly this thesis focuses on a centralized accelerated algorithm and explores the possibility of being used in distributed computing. We set up a suitable network structure for the algorithm to operate on with keeping at the same time the features of the accelerated algorithms. The original delayed algorithm is modified into an incremental algorithm that takes into consideration one agent's objective function at a time step. Specifically the computations at each time step are based on the objective function of the agent that is "active" at that time.

1.3 Outline

The thesis is organized as follows. In Chapter 2 we review the most important algorithms in centralized convex optimization. We investigate the necessary features of the algorithms that solve convex optimization problems bearing in mind that our motivation is to transfer these algorithms in the decentralized setting, where we want simple computations in each step and the minimum amount of communication through the agents. Also we are going to see the optimal bounds for classes of convex functions and the associated optimal algorithms.

In Chapter 3 we consider decentralized algorithms for convex optimization. We provide the network structures and a number of algorithms that work on each structure and present their unique ideas and convergence rates.

Chapter 4 offers an accelerated delayed distributed algorithm for decentralized convex optimization along with simulation results upon quadratic functions. We provide an algorithm in centralized setup that is able to handle outdated information in the form of past gradients and analyze a continuous time approximation of the delayed dynamics using differential equations. Next we focus in the decentralized setting as we modify the algorithm accordingly into an incremental accelerated algorithm to fit the master-server network structure.

Finally in Chapter 5 we present our conclusions for this area of research and provide directions for future work.

CHAPTER 2
Centralized first-order methods for convex optimization

First order methods have been heavily studied and have been proven very useful tools for solving centralized optimization problems. In this chapter we review of a number of those first order methods that will help us understand the reasons why these algorithms are needed in optimization with a perspective towards the distributed setting.

2.1 First order oracle methods

When solving a problem P, such as an optimization problem or an everyday problem like routing your way from work to home, we have plenty of numerical methods that we can use, but we want to find a scheme that is best suited for P. However such a task is almost impossible. Imagine for example a problem P that has an optimal value $x^* = 0$. Then the method with the optimal "performance" is the one that reports in every step $x^* = 0$. Thus there is no point in finding the best method to solve a concrete problem P. Instead we should focus on finding methods for classes of problems $F \supset P$. Indeed, usually numerical methods are developed to solve many different problems with similar characteristics. Then the problem becomes: find a method M for the problem class F. Therefore we can define the performance of M on F as its performance on the worst problem from F.

Since we are going to evaluate a method M for a problem class F it is safe to assume that the method doesn't have perfect information about the concrete problem P. It has only a description of class F. In this case we need someone to tell us information about the problem P at every step of the procedure M. In order to model this situation, it is convenient to introduce the notion of the oracle O. An oracle O is just a unit, which answers the successive questions of the method. The method M is trying to solve the problem P by collecting and handling the data.

In general, each problem can be included in different problem classes. For each problem we can also develop the different types of oracles. But if we fix F and O, then we fix a model of our problem P . In this case, it is natural to define the performance of M on $(F\ ;\ O)$ as its performance on the worst P_w from F [12].

Next let us see the main types of oracles that exist in optimization theory. For all the oracles the input is an $x \in R^n$ but the output differs:

- Zero-order oracle: outputs $f(x)$
- First-order oracle: outputs $f(x)$ and $f^{'}(x)$
- Second-order oracle: outputs $f(x)$, $f^{'}(x)$ and $f^{''}(x)$

where $f^{'}(x)$ and $f^{''}(x)$ is the first and second derivative of function f accordingly.

As can be seen the amount of information regarding the problem increases as the order of the oracle. This means that methods that use higher order oracles require more information and therefore they implement complex procedures in each iteration. But as mentioned in the introduction, the huge amount of information available and the need to make the data allocation decentralized guides us to use problem solving methods that in each iteration do not perform

load-heavy tasks. Thus in this thesis we will deal with First-order oracle methods for convex optimization problems.

2.2 Smooth convex functions

In this thesis and generally in the optimization field it is favorable to deal with function (problem) classes that have "nice" properties in order to make the optimization procedure faster and the convergence to an optimal value plausible with certainty. As before we are concerned with the unconstrained minimization problem of a differentiable function f

$$\min_{x \in R^n} f(x). \tag{2.1}$$

It is preferable then that the function f belongs to the class of convex or strongly convex differentiable functions. We denote the class of one time differentiable convex functions as F^1 and the class of one time differentiable strongly convex functions as S^1. We can continue to the definition of the convex function:

Definition 2.2.1 A continuously differentiable function $f(x)$ is called convex on R^n if for any $x, y \in R^n$ we have:

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle. \tag{2.2}$$

Examples of a convex function is the exponential function $f(x) = \exp(x)$ and the absolute value function $f(x) = |x|$.

In addition to the class of convex functions sometimes we prefer to impose something stronger as a restriction in order to get global convergence and faster

convergence rates. The following assumption for a function that is $f \in F^1(R^n)$ leads us to the class of strongly convex functions.

Assumption 2.2.1 There exists some constant $\mu > 0$ such that for any $\hat{x}$ with $\nabla f(\hat{x}) = 0$ we have

$$f(x) \geq f(\hat{x}) + \mu \frac{1}{2} \|x - \hat{x}\|.$$

Definition 2.2.2 A continuously differentiable function $f(x)$ is called strongly convex on R^n with notation $f \in S^1_\mu(R^n)$ if there exists a constant $\mu > 0$ such that for any $x, y \in R^n$ we have

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \mu \frac{1}{2} \|x - y\|. \tag{2.3}$$

An example of a strongly convex function is the quadratic function $f(x) = x^2$.

In the beginning of the section we also mentioned the notion of smoothness of the function. The smoothness notion is associated with the Lipschitz continuity of their respective gradients.

Definition 2.2.3 For a function to have Lipschitz continuous gradients with constant L in R^n means that for any $x, y \in R^n$ we have:

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|. \tag{2.4}$$

Examples of Lipschitz continuous functions are the absolute value function $f(x) = |x|$ and the sine function $f(x) = sin(x)$.

The notation for convex and strongly convex k times differentiable functions with Lipschitz continuous gradients and l times second order differentiable is $F_L^{k,l}$ and $S_{\mu,L}^{k,l}$ accordingly.

2.3 Lower bounds

Before we move forward and investigate the possible optimization methods for the classes of functions-problems that we have seen so far, it is useful to see the lower bounds for each class. By lower bounds we mean the lower complexity possible with regard to the iterations of the method in use, or in other words the number of oracle calls that are required from the method in order to achieve a solution close to the optimal value. Each of the two classes has different lower bounds and since for the strongly convex functions we have more restrictive assumptions we expect that the lower bounds to be better.

From Theorem 2.1.6 of [12] we have that the lower bounds for the class of convex functions $F_L^{\infty,1}(R^\infty)$ are

$$f(x_k) - f^* \geq \frac{3L\|x_0 - x^*\|}{32(k+1)^2}, \tag{2.5}$$

where $\{x_k\}$ is the sequence generated from the method, k is the time and x^* if the minimum of f with any initial point x_0.

Also from Theorem 2.1.12 of [12] we have that the lower bounds for the class of convex functions $S_{\mu,\mu Q_f}^{\infty,1}(R^\infty)$ are

$$f(x_k) - f^* \geq \frac{\mu}{2}\left(\frac{\sqrt{Q_f}-1}{\sqrt{Q_f}+1}\right)^{2k}\|x_0 - x^*\|, \tag{2.6}$$

where $\{x_k\}$ is the sequence generated from the method, k is the time,any constants $\mu > 0, Q_f > 1$ and x^* if the minimum of f with any initial point x_0.

2.4 Gradient descent

When we talk about techniques in optimization the first one that comes to mind is gradient descent. Gradient descent is easy to use and understand and provides a scalable solution to large optimization problems. The main idea behind this first order method is that the antigradient is the direction of the locally steepest descent of a differentiable function. The algorithm is described as follows:

Algorithm 1 Gradient Descent

1: $x_0 \in R^n$

2: **for** $k = 0$ to T **do**

3: $\quad x_{k+1} = x_k - h_k \nabla f(x_k)$

4: **end for**

5: **Output** : x_T

As simple as it seems this is the gradient descent. Obviously the only parameter is h_k which is called the step size of the gradient method and it is very important for the convergence procedure. Since we want to move in the antigradient direction the step size should be positive. The sequence $\{h_k\}$ is chosen in advance and there are many variants to the step size to choose from. Some of the important ones are:

- $h_k = h$
- $h_k = \frac{h}{\sqrt{k+1}}$
- $h_k = \underset{h \geq 0}{\operatorname{argmin}} f(x_k - h \nabla f(x_k))$

The simplest and the most heavily used is the constant step size. After analyzing the convergence rate through the improvement in each iteration we can consider the constant step size as the parameter of the expression and optimize accordingly in order to obtain the best rate possible. With this analysis we end up with the value $h_k = \frac{1}{L}$ for the class of convex functions and $h_k = \frac{2}{L+\mu}$ for the class of strongly convex functions, where L is the Lipschitz constant of the function. The resulting convergence rate as it is proven in [12] for the class of differentiable convex functions for $h_k = \frac{1}{L}$ is:

$$f(x_k) - f^* \leq \frac{2L\|x_0 - x^*\|}{k+4}, \tag{2.7}$$

and for the class of strongly convex differentiable functions for $h_k = \frac{2}{L+\mu}$ is:

$$f(x_k) - f^* \leq \frac{L}{2}\left(\frac{Q_f - 1}{Q_f + 1}\right)^{2k} \|x_0 - x^*\|^2, \tag{2.8}$$

where L is the Lipshitz coefficient, x^* is the optimal point, f^* is the optimal value of the function and $Q_f = \frac{L}{\mu}$ a constant.

Comparing the rate of convergence of the gradient method with the lower bounds for convex and strongly convex functions we can see that the method does not provide a convergence rate close to this lower bound. This will be the motivation to seek first order methods that achieve optimal convergence rates in further sections. On the other hand though, gradient descent is the bread and butter of optimization and it is a part of many other algorithms that transcend to the decentralized field.

2.5 Mirror descent

So far we have dealt with first order methods that attend to the unconstrained optimization problem of a convex function. Sometimes we want to constrain our solution in a subset of R^n due to practicality and feasibility reasons that we encounter in real life applications. Also mirror descent with mild alterations is used for decentralized solutions and provides a delay tolerant framework as we will see in the next chapter. Basically if we have a set $X \subset R^n$ which is assumed compact, the problem is set as:

$$\min_{x \in X} f(x). \tag{2.9}$$

Constraining the solution set affects algorithms designed for unconstrained optimization such as gradient descent and requires alterations in order for the converging sequence to reside in the solution space. For this reason we need to introduce the notion of the projection operator Π_X on the set X:

$$\Pi_X(x) = \operatorname*{argmin}_{y \in X} \|x - y\|. \tag{2.10}$$

An important Lemma that is proven useful in the analysis is Lemma 3.1 from [4] that states:

Let $x \in X$ and X is compact and convex and $y \in R^n$ then

$$(\Pi_X(y) - x)^T (\Pi_X(y) - y) \leq 0 \tag{2.11}$$

$$\|\Pi_X(y) - x\|^2 + \|\Pi_X(y) - y\|^2 \leq \|y - x\|^2. \tag{2.12}$$

The mirror descent algorithm is based on the idea of moving through the dual space of the gradient operators and projecting these points on the original solution space. In order to explain the method we have to point out that if the solution space is a Banach space $\mathbf{B}$ the gradients reside in the dual space $\mathbf{B}^*$. The great insight of Nemirovski and Yudin [11], that created mirror descent, is that one can still do gradient descent in the dual space by first mapping the point $x \in \mathbf{B}$ into the dual space $\mathbf{B}^*$, then performing a gradient descent step in the dual space, and finally mapping the point back to the primal space $\mathbf{B}$. This operation might result in a point that does not reside in the restricted solution space X, so we have to project the point back inside to this space.

Another useful notion is the Bregman Divergence associated with the function f and is defined as:

$$D_f(x, y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle, \tag{2.13}$$

and the main identity that follows this definition is

$$\langle \nabla f(x) - \nabla f(y), x - z \rangle = D_f(x, y) + D_f(z, x) - D_f(z, y). \tag{2.14}$$

In order to move from the primal space of the functions to the dual space of the gradients we need a mapping that will allow us do that, and this mapping is called a mirror map. Following the definitions from S. Bubeck's book [4], let $\mathbf{G} \subset R^n$ be a convex set such that X is included in its closure and $X \cap \mathbf{G} \neq \emptyset$. We say that $\Phi : \mathbf{G} \to R$ is a mirror map if it satisfies the following properties:

1. Φ is μ strongly convex and differentiable

2. The gradient of Φ takes all possible values

3. The gradient of Φ diverges on the boundary of $\mathbf{G}$

As mentioned previously the mirror map allows a point $x \in X \cap \mathbf{G}$ to be mapped to $\nabla\Phi(x)$ from where we can take a gradient step to get to $\nabla\Phi(x) - h\nabla f(x)$. Then the properties of the mirror map allows us to write $\nabla\Phi(y) = \nabla\Phi(x) - h\nabla f(x)$ for some $y \in \mathbf{G}$. The point y may reside outside of the solution space X so we have to project the point into that set. In Mirror Descent Bregman Divergence of the mirror map helps us to do that operation as:

$$\Pi_X^{\Phi}(y) = \operatorname*{argmin}_{y \in X \cap \mathbf{G}} D_{\Phi}(x, y). \tag{2.15}$$

Resulting from the definition of the mirror map and its properties, it is guaranteed the existence and the uniqueness of this projection. After all these definitions we can now proceed to the Mirror Descent method:

Algorithm 2 Mirror Descent

1: $x_1 \in \operatorname*{argmin}_{x \in X \cap \mathbf{G}} \Phi(x)$
2: **for** $k = 1$ to T **do**
3: $\quad \nabla\Phi(y_{k+1}) = \nabla\Phi(x_k) - h\nabla f(x_k)$
4: $\quad x_{k+1} \in \Pi_X^{\Phi}(y_{k+1})$
5: **end for**
6: **Output** : x_T

Or with the help of some algebra we end up with the more common view of the Mirror Descent that is seen as a definition most times (from Beck and Teboulle [2]) and gives a proximal point of view in the sense that the algorithm is trying to

minimize the local linearization of the function while not moving too far away from the previous point (with distances measured via the Bregman divergence of the mirror map).:

Algorithm 3 Mirror Descent

1: $x_1 \in \underset{x \in X \cap \mathbf{G}}{\operatorname{argmin}} \Phi(x)$
2: **for** $k = 1$ to T **do**
3: $\quad x_{k+1} = \underset{x \in X \cap \mathbf{G}}{\operatorname{argmin}}(h \nabla f(x_k) x + D_\Phi(x, x_k))$
4: **end for**
5: **Output** : x_T

Regarding the convergence rate of the Mirror Descent algorithm we have from Theorem 4.1 of [4] that, if $M^2 = \sup_{x \in X \cap \mathbf{G}} \Phi(x) - \Phi(x_1)$ and $h = \frac{M}{L}\sqrt{\frac{2\mu}{k}}$, then :

$$f\left(\frac{1}{k}\sum_{s=1}^{k} x_s\right) - f(x^*) \leq ML\sqrt{\frac{2}{\mu k}}. \tag{2.16}$$

2.6 Nesterov's accelerated algorithm

So far we have seen first order methods that have convergence rates that are far from optimal for the classes of convex and strongly convex functions. This means that the lower bounds that we saw in the previous section are not met by the convergence rate of these algorithms. Nesterov [12] introduced a pioneer algorithm that achieves the optimal convergence rate regarding the lower bounds for strongly convex differentiable functions. This algorithm is known as Nesterov's accelerated algorithm and relies on the notion of the estimate sequence. In gradient descent, the method forms a relaxation sequence that means that at

each step of the function the value of the function is getting smaller and smaller: $f(x_{k+1}) \leq f(x_k)$. Instead the estimate sequence is defined in [12] as:

Definition 2.6.1: A pair of sequences $\{\phi_k(x)\}_{k=0}^{\infty}$ and $\{\lambda_k\}_{k=0}^{\infty}$, $\lambda_k \geq 0$ is called an estimate sequence of a function $f(x)$ if $\lambda_k \to 0$ for any $x \in R^n$ and all $k \geq 0$ we have

$$\phi_k \leq (1 - \lambda_k)f(x) + \lambda_k \phi_0(x). \tag{2.17}$$

The point of this definition is made apparent in the following Lemma 2.2.1 from [12]: If for a sequence $\{x_k\}$ we have

$$f(x_k) \leq \phi_k^* = \min_{x \in R^n} \Phi(x), \tag{2.18}$$

then $f(x_k) - f^* \leq \lambda_k(\phi_0(x^*) - f^*) \to 0$ as $k \to \infty$.

As it can be seen, the convergence of the sequence $\{x_k\}$ that follows the estimate sequence depends solely on the convergence rate of the sequence $\lambda_k \to 0$ and hence at a first glance it simplifies things. But the tricky part is how to form the estimate sequence and how to ensure the Lemma 2.2.1 from [12]. Nesterov crafted a series of lemmas and theorems to answer these questions and provides the accelerated algorithm. Lemma 2.2.2 of [12] states that if

- f is an μ-strongly convex differentiable, L-Lipshitz function in R^n
- ϕ_0 is an arbitrary function on R^n
- $\{y_k\}_{k=0}^{\infty}$ is an arbitrary sequence on R^n
- $\{a_k\}_{k=0}^{\infty}$, $a_k \in (0, 1)$, $\sum_{k=0}^{\infty} a_k = \infty$
- $\lambda_0 = 1$

Then the pair of sequences $\{\phi_k(x)\}_{k=0}^{\infty}$, $\{\lambda\}_{k=0}^{\infty}$ defined by the following recursive rules:

$$\lambda_{k+1} = \lambda_k(1 - a_k), \tag{2.19}$$

$$\phi_{k+1} = (1 - a_k)\phi_k(x) + a_k(f(y_k) + \langle \nabla f(y_k), x - y_k \rangle + \frac{\mu}{2}\|x - y_k\|^2, \tag{2.20}$$

is an estimate sequence.

This Lemma places a framework around the estimate sequence and by taking a simple quadratic function as ϕ_0 Nesterov proceeds with the Lemma 2.2.3 in [12] that states that if $\phi_0 = \phi_0^* + \frac{\gamma_0}{2}\|x - v_0\|^2$ then

$$\phi_k(x) = \phi_k^* + \frac{\gamma_k}{2}\|x - v_k\|^2, \tag{2.21}$$

where

$$\gamma_{k+1} = (1 - a_k)\gamma_k + a_k\mu,$$

$$v_{k+1} = \frac{1}{\gamma_{k+1}}\left((1 - a_k)\gamma_k a_k + a_k\mu y_k - a_k\nabla f(y_k)\right),$$

$$\phi_{k+1}^* = (1-a_k)\phi_k + a_k f(y_k) - \frac{a_k^2}{2\gamma_{k+1}}\|\nabla f(y_k)\|^2 + \frac{a_k(1 - a_k)\gamma_k}{\gamma_{k+1}}(\frac{\mu}{2}\|y_k - v_k\|^2 + \langle \nabla f(y_k), v_k - y_k \rangle).$$

With these two lemmas we are led to the general scheme for Nesterov's accelerated method:

Algorithm 4 Nesterov's General Scheme

1: Choose $x \in R^n$, $\gamma_0 > 0$ and $v_0 = x_0$

2: **for** $k = 1$ to T **do**

3: Compute $a_k \in (0, 1)$ from $La_k^2 = (1 - a_k)\gamma_k + a_k\mu$

4: $\gamma_{k+1} = (1 - a_k)\gamma_k + a_k\mu$

5: $y_k = \frac{a_k\gamma_k v_k + \gamma_{k+1}x_k}{\gamma_k + a_k\mu}$

6: $x_{k+1} = y_k - \frac{1}{L}\nabla f(y_k)$

7: $v_{k+1} = \frac{1}{\gamma_{k+1}}\left((1 - a_k)\gamma_k v_k + a_k\mu y_k - a_k\nabla f(y_k)\right)$

8: **end for**

9: **Output** : x_T

We observe in this scheme that the gradient step is taken from the sequence $\{y_k\}$ and it is used to update the sequence $\{x_k\}$ which means that the auxiliary variable drives the sequence to the optimal point. Also as $k \to \infty$ and $x_k \to x^*$ the auxiliary variable gets closer and closer to the sequence $\{x_k\}$ since the difference between 2 consecutive points $x_{k+1} - x_k$ is diminishing as we can see in Algorithm 5 which is the commonly known Nesterov's accelerated algorithm for strongly convex functions.

Algorithm 5 Nesterov's Accelerated Method

1: $y_0 = x_0$

2: **for** $k = 1$ to T **do**

3: $\quad x_{k+1} = y_k - \frac{1}{L}\nabla f(y_k)$

4: $\quad y_{k+1} = x_{k+1} + \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(x_{k+1} - x_k)$

5: **end for**

6: **Output** : x_T

This form of the algorithm provides a clearer idea about the sequences entailed. There is the primary sequence $\{x_k\}$ that updates from a gradient step to meet a condition of getting lower values at each iteration and the auxiliary sequence $\{y_k\}$ that was stated arbitrary in the beginning but after the imposed restrictions for the estimate sequence to hold takes the form of a mixture of two different time steps of the sequence $\{x_k\}$. In this form of the algorithm it is assumed for simplicity that $a_0 = \sqrt{\frac{\mu}{L}}$. Many have tried to interpret how Nesterov's accelerated algorithm works with the help of differential equations [15] or using multiple optimization techniques to realize each variable of Nesterov's algorithm [16]. As for the convergence rate of the Nesterov's Accelerated algorithm Theorem 2.2.3 from [12] gives us:

$$f(x_k) - f^* \leq \min\left\{\left(1 - \sqrt{\frac{\mu}{L}}\right), \frac{4L}{(2\sqrt{L} + k\sqrt{\gamma_0})^2}\right\}(f(x_0) - f^* + \frac{\gamma_0}{2}\|x_0 - x^*\|^2), \tag{2.22}$$

where $\gamma_0 = \frac{a_0(a_0 L - \mu)}{1 - a_0}$

We can see that the convergence rate of this scheme coincides with the lower bounds set for the family of strongly convex differentiable functions, which makes the algorithm optimal for the type of class.

CHAPTER 3
Decentralized first order methods

Up until now we have seen methods that work in a centralized manner requiring all the information to be held in one machine and perform the computations locally. Nowadays the information is spread through the network and we want to take advantage of distributed solutions to speed up optimization procedures. Though usually the local speed up of the smaller optimization procedures is great, the amount of communication through the nodes in the network and the degree of synchronization that dictates the procedure makes the optimization process a challenging task. Also, as we will verify later in this section, the convergence rates of the decentralized algorithms are far from optimal and yet there is no algorithm that achieves the lower bounds for any class of functions with respect to the centralized setting. In this chapter we will examine network setups for distributed optimization and look into the literature for suitable algorithms for each setup and compare them.

Throughout the section we assume that there are n nodes in a network and that each one of them hold their own information-objective function $f_i(x)$ and the goal is to optimize the objective function of the network, which is the sum of all the agents objective functions.

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{3.1}$$

One key feature is the network topology and the way that the algorithms treat it. There are algorithms in decentralized convex optimization that neglect the network topology and work on a generic setting [9], [5], [7], [13] and others that take advantage of a static network topology to work on [14], [6], [1] . The first category relies upon the notion of consensus which means that each node communicates locally the information of its value and then aggregates the total information it received along with its value to obtain the next value. In the second category fall algorithms such as incremental algorithms and algorithms that work in the master-server setting. Both have a static network structure and the way that they approach the optimization problem relies on that structure. The shared idea is that the computations are made locally and the information is passed on from node to node for the procedure to carry on. Also another way to look at these two classes is that the first class does not require any knowledge of the network topology while the second class does.

As we have seen in this section there is a rise in interest in distributed optimization algorithms in recent years for convex optimization. Many of them are based on the seminal paper of Tsitsiklis et al [9], which sets the foundations for distributed asynchronous gradient optimization algorithms. These algorithms allow the agents to communicate locally with their neighbors and exchange information locally about their current values and each agent proceeds with the optimization procedure based on their information.

Since the results of [9] are rather important we present the main ideas of the article. Assume there is a connected graph $G = \{V, E\}$ for the underlying network

and $H_1, H_2...H_L$ finite dimensional vector spaces and $x = (x_1, x_2...x_L)$ with $x_l \in H_l$ being the $l - th$ component of x. Also let $\{1...M\}$ be the set of processors that participate in the distributed optimization process. At any time k processor i may transmit some (or all) of the components of $x^i(k)$ to some of the other processors. We assume that the communication delay is bounded. If a message from processor j of a component H_l is received by processor i at time k, let $t_l^{ij}(k)$ denote the time the message was sent. Therefore the content of the message is $x_l^j(t_l^{ij}(k))$. Once the processor i receives all the messages sent at time k then it proceeds to update his current value along with the computation of gradient step with regard to his own information. Therefore the updated value of the node is a convex combination of the received messages and his own gradient update. Then

$$x_l^i(k+1) = \sum_{j=1}^{M} a_l^{ij}(k) x_l^{ij}(t^{ij}(k)) + \gamma^i(k) s_l^i(k), \tag{3.2}$$

where $s_l^i(k)$ is the l-th component of $s^i(k)$ which is relative to the update rule and can be for example a gradient method:

$$s_i^i(k) = -\frac{\partial J}{\partial x_i} x^i(k), \tag{3.3}$$

and the coefficients are scalars satisfying:

1. $a_l^{ij}(k) \geq 0$
2. $\sum_{j=1}^{M} a_l^{ij}(k) = 1$
3. $a_l^{ij}(k) = 0$ when no message received or cannot be received due to lack of an edge in the graph

4. $\gamma^i(k)$ is the step size of the method being used, for example $\frac{1}{l}$ for gradient descent

We also observe that this algorithm is designed to handle delays. If $t^{ij}(k) \neq k$, meaning that the communication between nodes i and j is not instantaneous there will be delays in the transferred state of the neighboring node. In [9] the delays are assumed to be bounded, the transmission rate between two nodes is also bounded and there are numbers B, β such that there exists at least one message between i and j in the interval $[Bk^\beta, B(k+1)^\beta]$ and the number of transmissions is bounded in the interval. The presence of communication delays necessitates an augmented state if a state space representation is desired. Exploiting linearity, we conclude that there exist scalars $\Phi_l^{ij}(k|q)$, for $k \geq q$ such that:

$$x_l^i(k) = \sum_{j=1}^{M} \Phi_l^{ij}(k|0) x_l^j(1) + \sum_{q=1}^{k-1} \sum_{j=1}^{M} \gamma^j(q) \Phi_l^{ij}(k|q) s_l^j(q). \tag{3.4}$$

The coefficients $\Phi_l^{ij}(k|q)$ are determined by the sequence of transmission and reception times and the combining coefficients. Thus in general they are unknown, but they have properties that can be defining.

3.1 Consensus based distributed optimization algorithms

Other algorithms for distributed optimization in a static generic network setting rely on the separation of the objective function instead of the components of the feature vector. Again we assume a graph $G = \{V, E\}$ for the underlying network which is connected (and sparse), simple and undirected . In a generic network an important feature is the adjacency or weight matrix of the network. We assume that the weight matrix W is symmetric and doubly stochastic with

$\mu = \mu(W)$ the second largest singular value of W and R a bound on the distance on the solution. The work of Jakovetic et al in [7], [8] provides two accelerated algorithms for distributed optimization that operate in each iteration in way similar to our work. Both methods rely on the notion of consensus for converging to the average "truth" between nodes and it is an extension to the work of [9].

The first method in [8] is the Distributed Nesterov Gradient method (D-NG). This algorithm, much as Nesterov's for centralized optimization, generates a sequence $(x_i(k), y_i(k))$ at each node i where $y_i(k)$ is the auxiliary variable. Each agent performs updates after communicating with its neighbors and performing the gradient step with regard to his own value. The update equations are:

$$x_i(k+1) = \sum_{j \in O_i} W_{ij} y_j(k) - a_k \nabla f_i(y_i(k)) \tag{3.5}$$

$$y_i(k+1) = x_i(k+1) + \beta_k (x_i(k+1) - x_i(k)) \tag{3.6}$$

where W_{ij} are the weights of the weight matrix W for respecting nodes i and j and O_i is the neighborhood set of node i. The coefficients are defined as:

$$a_k = \frac{c}{k+1}, \qquad c > 0, \qquad \beta_k = \frac{k}{k+3} \tag{3.7}$$

At each iteration of the algorithm, each node broadcasts the auxiliary variable to its neighbors and receives their corresponding values. Then weighs each value- depending on how much he "trusts" the particular neighbor- and performs a negative gradient step with regard to its objective function f_i.

Another approach in [8] for an accelerated method in the generic network setting is to apply a consensus procedure in the two stages of the algorithm one for each sequence. The algorithm Distributed-Nesterov Consensus (D-NC) uses a constant step size $a < \frac{1}{2L}$ and operates in 2 scales. In the outer iteration k, each node updates $(x_i(k), y_i(k))$, just as D-NG, and in the inner iteration s the nodes perform 2 rounds of consensus for both sequences. The number of the inner iterations depend of the time k and the value of μ. Both the algorithms D-NG and D-NC will reach the optimal solution with the difference that the consensus on the first one is implicit and on the second one is explicit meaning that D-NG applies consensus on each sequence within the network and D-NC applies consensus on the pair. As we have seen so far k is the number of gradient evaluations for the algorithms. Let K be the number of per-node communications. The D-NG algorithm achieves convergence rate $O(\frac{logk}{k})$ and $O(\frac{logK}{K})$ where k is the per-node gradient evaluations and K is the per-node communications, since the communication occurs once in each iteration. The D-NC algorithm has less outer iterations and achieves a rate of $O(\frac{1}{k^2})$ and $O(\frac{1}{K^{2-\xi}})$ with $\xi > 0$ arbitrarily small. Generally the D-NG does not require any knowledge of the network and its structure and it is faster than D-NC on certain networks, especially with none complicated ones. The D-NC algorithm requires some knowledge of the network such as $\mu(W)$ and on the outer iterations behave much more like the centralized accelerated algorithm of Nesterov. Also the convergence rate depends on the weight of matrix W so it is harder to understand the convergence behavior.

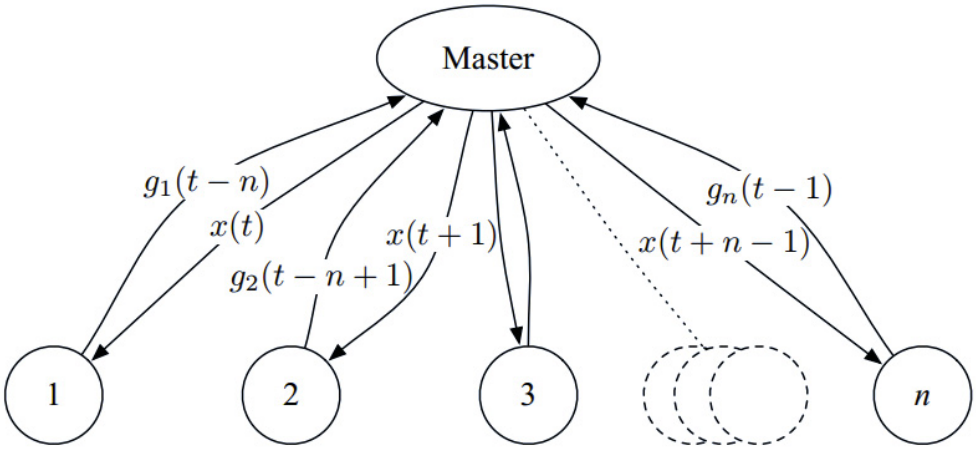

Figure 3–1: Figure 3.1: Cyclical Master-Server network setting where $g_i(x) = \nabla f_i(x)$ and the server nodes pass out-of-date information to the master with a delay of n [1]

3.2 Master-server based distributed optimization algorithm

Since our focus is distributed algorithms in a master-server network setting, Agarwal and Duchi in [1] present an elegant alteration to mirror descent that we have seen in the previous section and it is fitted for the needs of distributed convex optimization. Their approach handles the minimization of a stochastic function that samples over a data space that is distributed in the server nodes.

$$\min_{x \in X} f(x), f(x) = E\left[F(x, \xi)\right] \tag{3.8}$$

For this scheme, the algorithm handles gradients that are out of date and this means that instead of $g(t) = \nabla F(x(t), \xi)$ we get $g(t - \tau(t)) = \nabla F(x(t - \tau(t)), \xi)$ where $\tau(t)$ is the delay of time t and it is potentially random. In [1] they focus on two different algorithms: mirror descent and dual averaging. Since the mirror

descent method was presented in the previous chapter we will analyze only this method due to the fact that the analysis is similar for both methods.

As with most algorithms that we have seen before, the analysis of the distributed mirror descent requires the following assumptions

- **Assumption 1**: Lipshitz continuity which expands to bounded gradients of the function: $E\left[\|\nabla F(x,\xi)\|^2\right] \leq G^2$
- **Assumption 2**: Smoothness assumption-Lipschitz continuity of gradients and the variance bound $E[\|\nabla f(x) - \nabla F(x,\xi)\|_*^2] \leq \sigma^2$ holds
- **Assumption 3**: Compactness assumption: For $x^* \in \underset{x\in X}{\operatorname{argmin}} f(x)$ and $x \in X$ the bounds $\phi(x^*) \leq \frac{R^2}{2}$ and $D_\phi(x^*, x) \leq R^2$ both hold.

As we have previously seen the mirror descent algorithm is expressed as:

$$x(t+1) = \underset{x\in X}{\operatorname{argmin}} \left\{ \langle g(t), x\rangle + \frac{1}{a_t} D_\phi(x, x(t)) \right\}. \tag{3.9}$$

The idea of the authors of [1] is to implement the mirror descent algorithm as it is in a delayed decentralized setting of master-server network. Workers have the only responsibility of computing gradients based on the information they have and report them back to the master node when asked. The master node is the one performing the bulk of the algorithm. The master node at the start of each iteration selects a server node to exchange information. The selection could follow some deterministic rules like a cyclic rule going through the server nodes in a circle. Depending on the selection procedure the server node has information about the gradient of the function at the point where it last communicated with the master node. Of course the information that the server node has does not coincide

with the current information that the master node has. Assuming that each server has a delay $\tau(t)$ depending on the time stamp that it received the information and this delay is bounded by B, $E[\tau(t)] \leq B$, in the previous iterative equation we have to change the receiving gradient to $g(t - \tau(t))$ calculated in the point $x(t - \tau(t))$. The master node carries the mirror descent with the given gradient and returns the next point in the sequence x to the server node. Thus the delayed algorithm can be expressed as:

$$x(t+1) = \underset{x \in X}{\operatorname{argmin}} \left\{ \langle g(t - \tau(t)), x \rangle + \frac{1}{a_t} D_\phi(x, x(t)) \right\}. \tag{3.10}$$

Besides all the merits of decentralized algorithms though, we have to verify if it is worth transitioning from an algorithm that has proven itself in the centralized field to a distributed one in the master-server setting in the sense of faster overall convergence and robustness. Even though the master-server setting is not completely decentralized since it has a static network topology it maintains the main features such as distributed information to the agents and scalability through asynchronicity. In most functions small pertubations of the point calculating the gradient lead to big differences in the actual values. The Lipschitz continuity assumptions of the function and the gradients help towards robustness of pertubations since they upper bound that variation. But inherently we expect an error in the convergence rate of the algorithm due to the delay introduced in it. The goal of each algorithm that deals with these matters of asynchronicity is to restrain these terms that contain the delay into orders that converge asymptotically faster than the terms of the original algorithm.

Regarding the convergence rate of the delayed stochastic mirror descent in [1] it is proven that indeed the term of the delay is of a second order and asymptotically negligible. If we set $a(t) = \frac{1}{L+\eta(t)}$ and $\eta(t)$ is chosen to control the effects of delays and errors from the delayed stochastic information then from Theorem 2 of [1] and if we are under the assumption that the delay is constant as for example in the cyclical delayed master-server network, we obtain the convergence rate:

$$E[\sum_{t=1}^{T} f(x(t+1))] - Tf(x^*) \leq 2LR^2 + R^2(\eta(1)+\eta(T)) + \frac{\sigma}{2}\sum_{t=1}^{T}\frac{1}{\eta(t)} + 2LG^2(\tau+1)^2\sum_{t=1}^{T}\frac{1}{\eta(t-\tau)^2} + 2\tau GR. \tag{3.11}$$

If we optimize with regard to the convergence rate we get a value for $\eta = \frac{\sigma\sqrt{t+\tau}}{R}$ which leads to the bound

$$E[f(x(T))] - f(x^*) = O\left(\frac{LR^2+\tau GR}{T} + \frac{\sigma R}{\sqrt{T}} + \frac{LG^2\tau R2log(T)}{\sigma^2 T}\right). \tag{3.12}$$

The convergence rate is of the same order for time variant bounded delays that applies to the same network architecture. Although this is a robust algorithm and rather simple to implement and understand, the idea behind it is not optimal regarding the lower bounds that we have seen in the previous section for the classes of smooth convex and strongly convex functions. Also there is a obvious restraint regarding the size of the network since the convergence rate naturally comes from it and it can be clearly seen in the case of the cyclical master-server setting where the delay is equal to the server nodes of the network n.

3.3 Incremental gradient descent algorithms

In this section, as well in this chapter, we continue to analyze algorithms that are based in the master-server network setup mainly because we want to keep the idea of maintaining the information spread to the network and communicating gradients in a simple way. We have seen so far that this type of network inherently introduces delays in the queue of the process. Also network latency and work load in the server nodes could vary in a real scenario having an impact in the delay mechanism also. So it is important that a distributed algorithm has the attribute to respond to network delays accordingly and incorporate them as part of the process. Furthermore, another challenge in this formulation is to balance the delay penalty of waiting to collect gradients from all machines (to compute the gradient of the total loss) and the bias that occurs when the decision vector is updated incrementally as soon as new partial gradient information from a machine arrives [14]. In this section we give emphasis to the second setup which is called as an incremental approach. It is called like that since at each step of the procedure the algorithm does computations regarding partial derivatives without wasting time waiting to aggregate the sum of the whole.

Throughout the section as with most algorithms we have seen so far we make the following assumptions:

- **Assumption 1**: The objective function is smooth: $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$
- **Assumption 2**: The objective function is μ-strongly convex: $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|x - y\|^2$

So far from the algorithms of the centralized field the simplest and one of the most widely used is gradient descent. One idea would be to try and implement simple delays in a centralized manner in the beginning to gain some insight about how it could possibly work in the master-server setting. The iteration for the gradient descent for strongly convex functions would be:

$$x(k+1) = x(k) - a\nabla f(x(k)), \tag{3.13}$$

where $a = \frac{2}{\mu+L}$ for the strongly convex case [12]. Now lets assume that there is some delay introduced in each iteration regarding the computation of the gradients discarding the distribution of the function into partial functions giving them as objective functions to the server nodes. Then the gradient descent would take the shape of:

$$x(k+1) = x(k) - a\nabla f(x(k-\tau(k))), \tag{3.14}$$

where $\tau(k)$ is the delay of each step as a random number that $\tau : N \to N_0$. Imagine this artificial delay as the delay from a node in the master server network. One could say that this iterative procedure doesn't make much sense in the way that gradient descent was introduced. The philosophy behind the centralized algorithm was that we choose a point of a function and move towards the steepest descent which is the anti-gradient. But in that context the delayed algorithm fails to give such an intent. The iterations take a point of the function and move it towards the anti-gradient of a previous point. This could mean that the algorithm under the necessary assumptions would not even converge into the neighborhood of the

optimal point. More accurately the algorithm demands a function in which "small" pertubations around a point have a "small" impact in the gradients.

Thinking about the main goal of the gradient descent it is more logical to keep the same mindset when the delays are introduced. Therefore when a delayed gradient comes to be processed form the algorithm it could be paired with the matching point of the function to keep the same premises of the gradient descent. Though it is a bit unintuitive at first to keep back tracking to find the pair instead of using the most recent point of the sequence $\{x_k\}$ it is surprisingly more easy to analyze since the idea of the gradient descent remains the same. The main update rule would be:

$$x(k+1) = x(k-\tau(k)) - a\nabla f(x(k-\tau(k))). \tag{3.15}$$

We complete a gradient step at the delayed point that, in the sense of the master-server network, the master received by the worker. Each agent in the network has its own delayed information that the master passed to him in a previous moment and computes the gradient based on this information. Along with the "feeling" that this iterative procedure would probably converge, in Proposition 1 of [6] it states that if f is an μ-strongly convex function in R^n and the delay is bounded, $0 \leq \tau(k) \leq \tau_{max}$ and the optimal step size of $a = \frac{2}{\mu+L}$ then the sequence $\{x_k\}$ obeys:

$$\|x(k) - x^*\| \leq \left(\frac{Q-1}{Q+1}\right)^{\frac{k}{1+\tau_{max}}}, \tag{3.16}$$

where $Q = \frac{L}{\mu}$. One thing to note also is that the step size is independent of the delays and the convergence rate depends on the upper bound of the delay which could have some implications regarding the size of the network in the master-server setting.

In order to implement the same idea to the distributed delayed network of master-server we have some extra points to cover. First the function should be able to be written as:

$$f(x) = \sum_{i=1}^{M} f_i(x), \tag{3.17}$$

where each $f_i(x)$ is strongly convex and it is assigned as the objective function of each server. The setting for random delays implies that at every iteration of the algorithm a node that has completed the processing of its information is picked in a stochastic order. With these points in mind the authors of [6] proceed to the incremental algorithm:

Algorithm 6 Delayed Proximal Gradient Method

1: $x_0 \in R^n$

2: **for** $k = 0$ to T **do**

3: $\quad i(k) = U[1, M]$

4: $\quad s(k) = x(k - \tau(k)) - a\nabla f_{i(k)}(x(k - \tau(k)))$

5: $\quad x(k + 1) = (1 - \theta)x(k) + \theta s(k)$

6: **end for**

7: **Output** : x_T

where $\theta \in (0, 1]$ and $i(t)$ is being drawn from the uniform discrete distribution with support 1,2..M. Also the assumption made previously are in power with each $f_i(x)$ being L_i-smooth and $L \leq L_{max}$ with

$$L_{max} = \max_{1 \leq m \leq M} L_m.$$

The main theorem of [6] provides the convergence rate of the algorithm and states that:

Theorem 2 of [6]:If $\tau(k) \in [0, \tau_{max}]$ for all $k \in N$ and that the step size a satisfies

$$a \in \left(0, \frac{\mu}{L_{max}^2}\right),$$

then the sequence $\{x_k\}$ generated by the Delayed Proximal Gradient Method satisfies

$$E_{k-1}[f(x(k))] - f^* \leq \rho^k (f(x(0) - f^*) + \epsilon, \tag{3.18}$$

where E_{k-1} is the is the expectation over all random variables $i(0), i(1)..i(k-1)$, f^* is the optimal value of problem ,

$$\rho = \left(1 - 2a\mu\theta\left(1 - a\frac{L_{max}^2}{\mu}\right)\right)^{\frac{1}{1+\tau_{max}}}, \tag{3.19}$$

and

$$\epsilon = \frac{aL_{max}}{2M(\mu - aL_{max}^2)} \sum_{m=1}^{M} \|\nabla f_m(x^*)\|^2. \tag{3.20}$$

Theorem 2 shows that even with a constant step size, which can be chosen independently of the maximum delay bound τ_{max} and the number of objective

function components M , the iterations generated by the Delayed Proximal Gradient Method converge linearly to within a neighborhood of the optimal value. Note the inherent trade-off between ρ and ϵ: a smaller step-size a yields a smaller residual error ϵ but also a larger convergence factor ρ [6]. The algorithm provides descent convergence rate for the class of strongly convex functions but not nearly optimal, especially if we consider the tradeoff for the convergence neighborhood.

CHAPTER 4
An accelerated algorithm for delayed distributed convex optimization

In the previous chapters we have seen centralized and decentralized algorithms for convex optimization and every one of them has its pros and cons. We described centralized algorithms in order to realize and understanding of the existing methods we reviewed the most relevant ones that can take the leap to distributed optimization. In chapter 3 there was a discussion about decentralized algorithms. The main flaw of these methods is, as we have seen, their convergence rates. The convergence rates of distributed methods are far from optimal and also they may guarantee a convergence within a neighborhood of the optimal point without the decrease of this neighborhood in time. This is reasonable if someone takes into consideration the errors that the delays introduce in the network of the agents. The main goal of this thesis and especially this chapter is to provide an algorithm that converges with an optimal convergence rate for the class of functions it refers to with the error introduced by the delay to be negligible.

For these reasons we turn to marry the benefits of accelerated algorithms of the centralized field and successful decentralized techniques. The main issue remains the stability of the algorithm when delays are introduced and also their impact on the convergence rate. The focus of this chapter is to use an accelerated setup for the decentralized algorithm and borrow insight from the incremental algorithms. It is obvious why we are interested in accelerated algorithms since

they are the only known algorithms that have optimal convergence rates for the class of functions that are of interest. Although introducing delays into these algorithms is trickier than the common methods like gradient descent because of the auxiliary sequence $\{y_k\}$. This difficulty we try to offset with the insight we have by investigating incremental algorithms.

4.1 Differential equation for modeling Nesterov's accelerated algorithm with delay

Since the analysis of Nesterov's accelerated scheme with delays is challenging we select to analyze the scheme as a centralized algorithm that has inherent delay in its gradients. Nesterov's accelerated algorithm can be written as:

Algorithm 7 Nesterov's Accelerated Method

1: $y_0 = x_0$

2: **for** $k = 1$ to T **do**

3: $\quad x_{k+1} = y_k - s\nabla f(y_k)$

4: $\quad y_{k+1} = x_{k+1} + \frac{k-1}{k+2}(x_{k+1} - x_k)$

5: **end for**

6: **Output** : x_T

and the momentum coefficient of $(x_{k+1} - x_k)$ can be written as $\frac{k-1}{k+2} \approx 1 - \frac{3}{k}$. Our goal is to introduce delays in the gradients as if the gradients are taken from an asynchronous decentralized system in a master-server network. Let τ be a constant delay that is introduced into the gradients that the processing unit

receives. Then the first statement of the algorithm should change accordingly:

$$x_{k+1} = y_k - s\nabla f(y_{k-\tau}). \tag{4.1}$$

The delay that is inserted it is assumed constant-imagine something like a cyclical master-server setting-for the convenience of the analysis. Also further assumptions that are made are the same as in Nesterov's algorithm:

- **Assumption 1**: The function is μ-strongly convex
- **Assumption 2**: The function is L-smooth

From here we observe that we inserted a delay in a gradient step iteration. From the analysis in the section of incremental distributed algorithms we have seen that gradient steps behave smoothly and are easier to analyze if the gradient iteration is taken with regard to one point such as $y_{k-\tau}$ and not two points as $y_{k-\tau}$ and y_k. Thus this intuition urge us to write the update step as:

$$x_{k+1} = y_{k-\tau} - s\nabla f(y_{k-\tau}). \tag{4.2}$$

Also since the x_{k+1} is a gradient update with respect to $y_{k-\tau}$ we want to be consistent towards the update of y_{k+1} and replace the difference term $(x_{k+1} - x_k$ with $(x_{k+1} - x_{k-\tau}$. This difference originally was a difference between the updated x value and the value of x of the time step that the update was made so he have to honor this logic in addition to the fact that it helps with the analysis of the algorithm. Rewriting the algorithm and taking the momentum coefficient as a parameter:

Algorithm 8 Nesterov's Accelerated Method with Delay

1: $y_0 = y_1 = .. = y_{k-\tau} = x_{k-\tau} = .. = x_0$

2: **for** $k = \tau + 1$ to T **do**

3: $\quad x_{k+1} = y_{k-\tau} - s\nabla f(y_{k-\tau})$

4: $\quad y_{k+1} = x_{k+1} + \beta_{k+1}(x_{k+1} - x_{k-\tau})$

5: **end for**

6: **Output** : x_T

In order to analyze this accelerated algorithm we draw inspiration from the differential analysis of Nesterov's algorithm in [15]. First we combine the equations of the main loop:

$$x_{k+1} = y_{k-\tau} - s\nabla f(y_{k-\tau}) = x_{k-\tau} + \beta_{k-\tau}(x_{k-\tau} - x_{k-2\tau-1}) - s\nabla f(y_{k-\tau})$$

$$\frac{x_{k+1} - x_{k-\tau}}{\sqrt{s}} = \frac{\beta_{k-\tau}}{\sqrt{s}}(x_{k-\tau} - x_{k-2\tau-1}) - \sqrt{s}\nabla f(y_{k-\tau}).$$

We introduce $x_k = X(k\sqrt{s})$ for some smooth curve of $X(t)$ and define the delay as $\hat{\tau} = \tau\sqrt{s}$. The Taylor expansion with these approximations is:

$$\frac{x_{k+1} - x_{k-\tau}}{\sqrt{s}} = \dot{X}(t - \hat{\tau})(1 + \tau) + \frac{1}{2}\ddot{X}(t - \hat{\tau})\sqrt{s}(1 + \tau)^2 + o(\sqrt{s}),$$

$$\frac{(x_{k-\tau} - x_{k-2\tau-1})}{\sqrt{s}} = \dot{X}(t - \hat{\tau})(1 + \tau) - \frac{1}{2}\ddot{X}(t - \hat{\tau})\sqrt{s}(1 + \tau)^2 + o(\sqrt{s}),$$

$$\sqrt{s}\nabla f(y_{k-\tau}) = \sqrt{s}\nabla f(X(t - \hat{\tau})) + o(\sqrt{s}).$$

Where we use that $y_{k-\tau} - X(t-\hat{\tau}) = o(1)$ due to the Lipschitz continuity of function f. Next we define the value of the momentum coefficient such that

$$\beta_k = 1 - \frac{\hat{\beta}}{k} \Rightarrow \beta_t = 1 - \frac{\hat{\beta}\sqrt{s}}{t},$$

where $\hat{\beta}$ is a constant. And placing them all together

$$\begin{aligned}
&\dot{X}(t-\hat{\tau})(1+\tau) + \frac{1}{2}\ddot{X}(t-\hat{\tau})\sqrt{s}(1+\tau)^2 + o(\sqrt{s}) = \\
&\left(1 - \frac{\hat{\beta}\sqrt{s}}{t-\tau}\right)\left(\dot{X}(t-\hat{\tau})(1+\tau) - \frac{1}{2}\ddot{X}(t-\hat{\tau})(1+\tau)^2\sqrt{s} + o(\sqrt{s})\right) \\
&- \sqrt{s}\nabla f(X(t-\hat{\tau})) + o(\sqrt{s}). \qquad (4.3)
\end{aligned}$$

And by comparing coefficients of $\sqrt{s}$ since all the other terms cancel or are of higher order

$$\ddot{X}(t-\hat{\tau}) + \frac{\hat{\beta}}{(t-\tau)(1+\tau)}\dot{X}(t-\hat{\tau}) + \frac{1}{(1+\tau)^2}\nabla f(X(t-\hat{\tau})) = 0. \qquad (4.4)$$

We can define $t - \hat{\tau} = t^{'}$ and $\alpha = \frac{\hat{\beta}}{1+\tau}$ the differential equation will be

$$\ddot{X} + \frac{\alpha}{t^{'}}\dot{X} + \frac{1}{(1+\tau)^2}\nabla f(X) = 0, \qquad (4.5)$$

with initial conditions $X(0) = x_0$ and $\dot{X}(0) = 0$. The convergence of the ODE is guaranteed by theorem 2.1 of [15] that states for the same ODE: for any $f \in F_\infty \triangleq \cup_{L>0} F_L$ and any $x_0 \in R^n$ the ODE with initial conditions $X(0) = x_0$ and $\dot{X}(0) = 0$ has a unique global solution X.

4.2 Convergence analysis

We have seen that we can analyze this new accelerated delayed algorithm with differential equations. The question remains if the convergence rate of the accelerated algorithm is preserved and it is answered in theorem 4.1.

Theorem 4.1. For any convex f let $X(t)$ be the unique solution to the ODE with initial conditions $X(0) = x_0$ and $\dot{X}(0) = 0$ and $t > 0$. Then:

$$f(X(t)) - f^* \leq \frac{(\alpha-1)^2(1+\tau)^2\|x_0 - x^*\|^2}{2t^2}. \tag{4.6}$$

Proof

Define the energy functional

$$E(t^{'}) = \frac{2}{(\alpha-1)(1+\tau)^2}(t^{'})^2(f(X(t^{'})) - f^*) + (\alpha-1)\|X(t^{'}) + \frac{t^{'}}{\alpha-1}\dot{X}(t^{'}) - X^*\|^2 \tag{4.7}$$

$$\begin{aligned}\dot{E}(t') = &\frac{4}{(\alpha-1)(1+\tau)^2}(t^{'})(f(X(t^{'})) - f^*) + \frac{2}{(\alpha-1)(1+\tau)^2}(t^{'})^2\langle\nabla f, \dot{X}\rangle \\ &+ 2(\alpha-1)\langle X(t^{'}) + \frac{t^{'}}{\alpha-1}\dot{X}(t^{'}) - X^*, \frac{\alpha}{\alpha-1}\dot{X} + \frac{t^{'}}{\alpha-1}\ddot{X}\rangle.\end{aligned} \tag{4.8}$$

From the definition of our differential equation we have:

$$\frac{\alpha}{\alpha-1}\dot{X} + \frac{t^{'}}{\alpha-1}\ddot{X} = \frac{t^{'}}{\alpha-1}\left(\ddot{X} + \frac{\alpha}{t^{'}}\dot{X}\right) = \frac{t^{'}}{(\alpha-1)(1+\tau)^2}\nabla f.$$

Then

$$\begin{aligned}\dot{E}(t') &= \frac{4}{(\alpha-1)(1+\tau)^2}(t')(f(X(t'))-f^*) + \frac{2}{(\alpha-1)(1+\tau)^2}(t')^2\langle\nabla f, \dot{X}\rangle \\ &+ 2(\alpha-1)\langle X(t') + \frac{t'}{\alpha-1}\dot{X}(t') - X^*, \frac{t'}{(\alpha-1)(1+\tau)^2}\nabla f\rangle \\ &= \frac{4}{(\alpha-1)(1+\tau)^2}(t')(f(X(t')-f^*) + \frac{2(\alpha-1)t'}{(\alpha-1)(1+\tau)^2}\langle X(t') - X^*, \nabla f\rangle.\end{aligned} \tag{4.9}$$

Using the assumption of convexity of f:

$$\begin{aligned}\dot{E}(t') &\leq \frac{4}{(\alpha-1)(1+\tau)^2}(t')(f(X(t'))-f^*) + \frac{2t'}{(1+\tau)^2}(f(X(t')-f^*) \\ &= \frac{2t'(3-\alpha)}{(1+\tau)^2(\alpha-1)}(f(X(t'))-f^*) \leq 0,\end{aligned} \tag{4.10}$$

for $\alpha > 3 \Rightarrow \beta > 3(1+\tau)$.

Since we know that $f(X) \geq f^*$ the energy functional is non increasing and the term $(\alpha-1)\|X(t') + \frac{t'}{\alpha-1}\dot{X}(t') - X^*\|^2$ is positive by definition we can deduce

$$\frac{2(t')^2}{(\alpha-1)(1+\tau)^2}(f(X(t'))-f^*) \leq E(t') \leq E(0) = (\alpha-1)\|x_0 - x^*\|^2.$$

Rearranging terms gives:

$$f(X(t')) - f^* \leq \frac{(\alpha-1)^2(1+\tau)^2\|x_0 - x^*\|^2}{2(t')^2}. \tag{4.11}$$

4.3 Incremental accelerated algorithm for the master-server setting

Once again we turn our attention to the master-server setting and the minimization of the objective function:

$$f(x) = \sum_{i=1}^{M} f_i(x) \tag{4.12}$$

In order to push this accelerated algorithm to the distributed master-server setting a few minor changes should be made and the idea is similar to the one of the incremental distributed algorithms [6]. First we have to take into consideration that each agent has its oracle that calculates the gradient of and send this information back to the master node in an asynchronous manner. We make the usual assumptions that:

- **Assumption 1**: The objective function is smooth: $\|\nabla f_i(x) - \nabla f_i(y)\| \leq L_i\|x - y\| \qquad \forall i \in \{1..M\}$
- **Assumption 2**: The objective function is μ-strongly convex: $f(y) \geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\mu}{2}\|x - y\|^2$

We want our algorithm to be able to handle delays so it is appropriate to maintain most of the accelerated algorithm presented in the previous section. The difference is that we assume that the master node selects a node from the available workers in random to share information and thus the delay introduced in the gradients is random but bounded $\tau(k) \leq B$ and the gradient is calculated for a portion of f which corresponds to the certain node f_i. We desire to create an incremental algorithm as in [6] fitting the delayed accelerated algorithm of the previous section.

Combining these two ideas in an incremental accelerated algorithm we come up with:

Algorithm 9 Incremental accelerated algorithm for the master-server setting

1: $y_0 = x_0$

2: **for** $k = 1$ to T **do**

3: $\quad i(k) = U[1, M]$

4: $\quad x_{k+1} = y_{k-\tau(k)} - s\nabla f_{i(k)}(y_{k-\tau(k)})$

5: $\quad y_{k+1} = x_{k+1} + \beta_{k+1}(x_{k+1} - x_{k-\tau(k)})$

6: **end for**

7: **Output** : x_T

where $U[1, M]$ is a uniform discrete distribution in [1,M] and $\tau(k)$ is the delay at time k.

4.4 Experimental results

To evaluate the performance of the proposed algorithms we have focused on unconstrained quadratic programming optimization problems because of their frequent appearance in applications such as portfolio optimization, regression and decision analysis, and because of the knowledge of the exact solution in closed form. We are interested in solving optimization problems of the form :

$$f(x) = \frac{1}{2}x^T Px + qx \tag{4.13}$$

and for the incremental accelerated algorithm for the master-server setting

$$f(x) = \frac{1}{M}\sum_{m=1}^{M}\left(\frac{1}{2}x^T P_m x + q_m x\right) \tag{4.14}$$

We have chosen to use randomly generated instances, where the matrices P_m and the vectors q_m are generated as explained in [10]. We choose the size of the network to be a variable in order to show how the general expected delay upper bound affects the convergence rate. In figure 4–1 we show the convergence rate of Nesterov's accelerated algorithm with delay of 3,7 and 15 respectively in a single node. There is no surprise that when the delay goes up the convergence rate slows down something that is aligned with the theoretical background that we provided. We performed 500 trials and we show the logarithmic average of all the tests. Also we observe a higher oscillation when the delay is increased. This happens due to the fact that an older value that is further away from the optimal is called back in the algorithm due to the higher delay and takes more time to wash this value away.

Figure 4–2 corresponds to experiments with the incremental accelerated algorithm for the master-server setting with 3, 7 and 15 nodes respectively. In these results we observe, in a fashion similar as with Nesterov's algorithm with delays, that the size of the network and thus the expected delay impacts the convergence rate. Also we can see that the convergence is in a neighborhood of the optimal point with some residual error ϵ such as in [14]. Also we observe that as the network becomes bigger and the expected delay higher the neighborhood of convergence gets larger which means larger residual errors. From the simulations we derive that the convergence rate of the incremental accelerated algorithm is faster by quite a margin from decentralized mirror descent [1] for the master

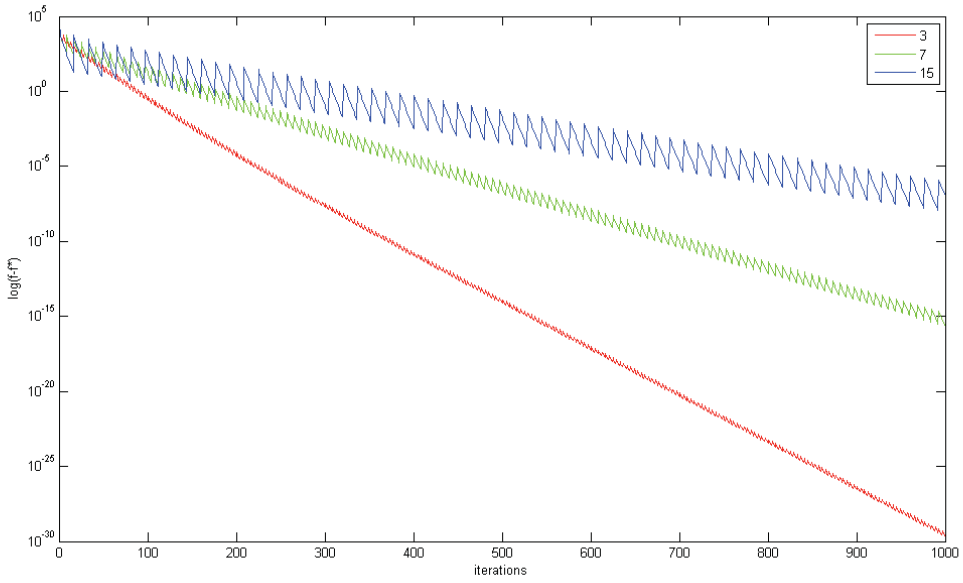

Figure 4–1: Logarithmic error of Nesterov's accelerated algorithm with delay = 3,7,15

server setting since the plots are showing at least linear convergence. As before we performed 500 trials and averaged the results.

In figures 4–3, 4–4 and 4–5 we compare the accelerated incremental algorithm with the incremental proximal gradient method from [6]. In each iteration one node updates the gradient value for the master server uniformly at random and the master returns the current value. The trials were repeated 500 times and we show the average of the results of the error. For this experiment we used a constant momentum coefficient small enough to be close to the corresponding coefficient of the incremental method. The incremental proximal gradient method has a linear convergence rate for the class of strongly convex functions. As we can see from the figures the accelerated incremental method appears to

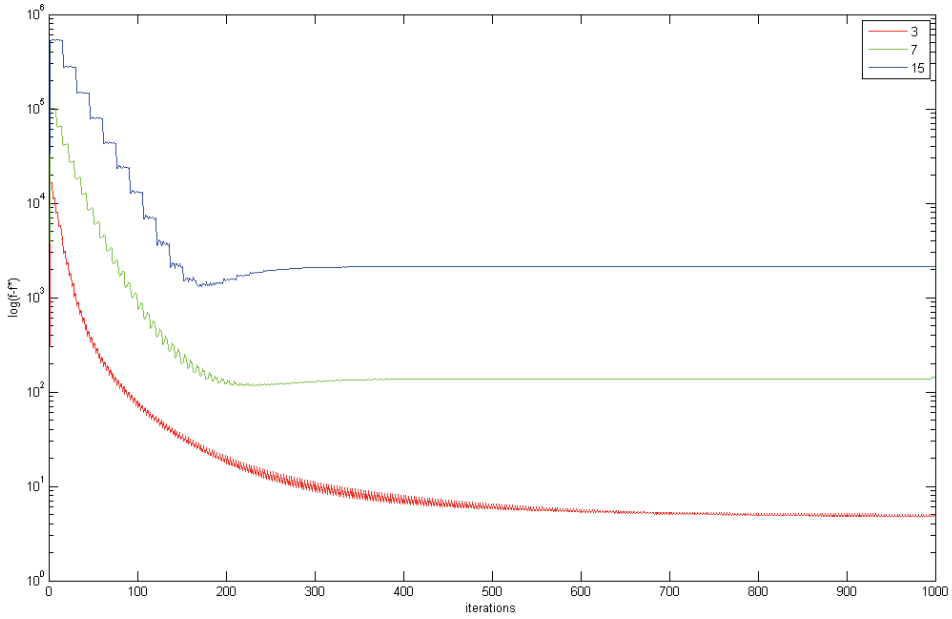

Figure 4–2: Logarithmic error of the incremental accelerated algorithm for the master-server setting with 3, 7 and 15 nodes

converge faster to the neighborhood of the optimal solution demonstrating, at least experimentally, that the convergence rate is at least linear with a factor of equal or less than ρ that we have seen in section 3.2.

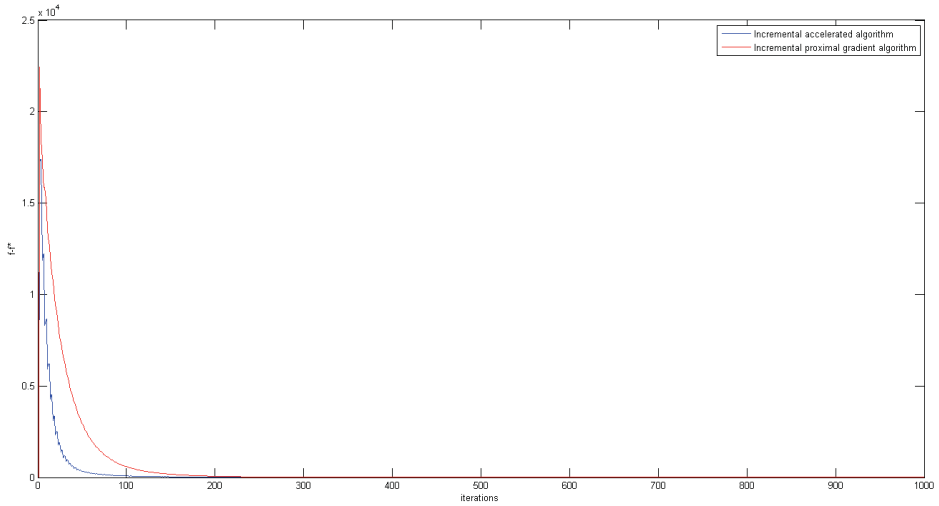

Figure 4–3: Comparison of error of the incremental accelerated algorithm and the incremental proximal gradient method for the master-server setting with 3 nodes

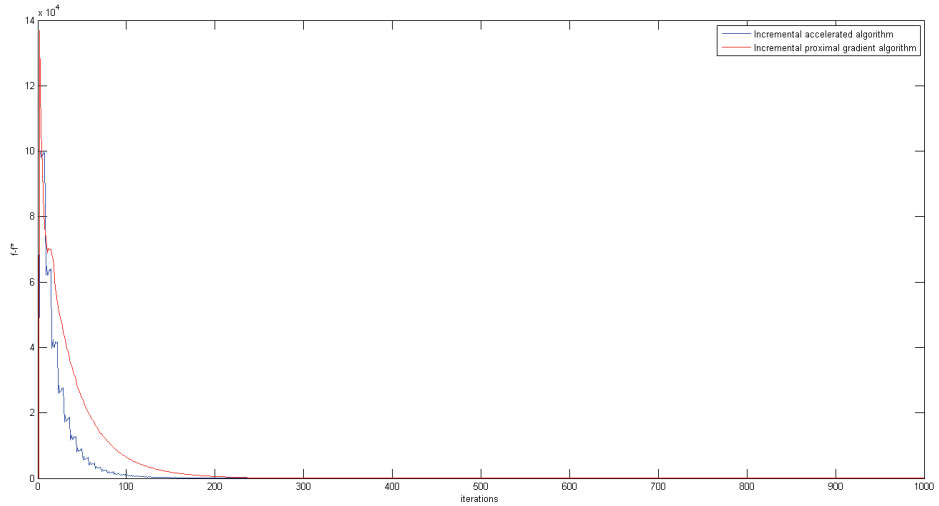

Figure 4–4: Comparison of error of the incremental accelerated algorithm and the incremental proximal gradient method for the master-server setting with 7 nodes

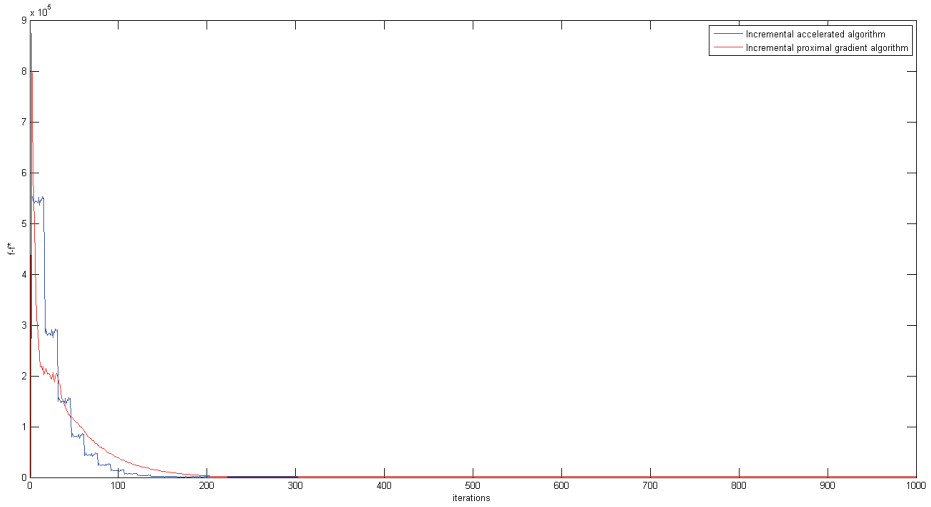

Figure 4–5: Comparison of error of the incremental accelerated algorithm and the incremental proximal gradient method for the master-server setting with 15 nodes

CHAPTER 5
Conclusion

5.1 Summary

In this thesis, we have examined centralized and decentralized algorithms for convex optimization and observed their strengths and weaknesses. From all that we have seen we verified that the transition from centralized solutions to decentralized ones is not simple and requires further assumptions and network setups. Simple methods like gradient descent are more suited in large scale networks since the simplicity of each iteration allows the agent to focus on the communication overhead and not on its own computations. Also we have seen that it is especially difficult to manage delays in distributed asynchronous convex optimization and plug and play solutions like the mirror descent which requires static setups like the master-server setting. The delays and the further assumptions lead to the fact that up until now there is no optimal algorithm for the class of convex and strongly convex functions that resides in the decentralized field.

Our contribution focuses on the creation of an accelerated algorithm for the master-server network setup with a convergence rate that is optimal for the class of convex and strongly convex functions. Our aim was to merge the optimal algorithm for strongly convex functions of the centralized field with techniques that are used in the decentralized algorithms using at the same time a

static, star network setup. We managed to provide theoretical proof for constant delay accelerated algorithm and a framework for distributed delayed convex optimization. Also the simulation results coincide with the theoretical ones and provide promising results for the distributed case of master-server network setting.

This thesis also provides insight about the common features of known centralized algorithms that can work in a decentralized environment. The key for these algorithms to work is to configure the update step to consider the delayed value instead for the current value for the computations. For example in mirror descent the update takes place in the dual space of the gradients where the update time is set to the delayed time stamp by default. In incremental gradient descent the whole gradient step is rolled back to the time $k - \tau(k)$. Also in the accelerated algorithm that we introduced the gradient step is rolled back in time and the auxiliary variable is based around $k - \tau(k)$ time stamp. In other words the known centralized algorithms have to take into account the delay of the gradients and incorporate them in their respective update steps.

5.2 Future research

According to our contribution the most needed result that is absent from our research is the theoretical proof for the incremental accelerated algorithm for the master-server setting. That would set a cornerstone for future endeavors in the distributed optimization field and provide a framework for future accelerated algorithms for different classes of functions in order for the decentralized algorithms to achieve optimal convergence rates.

Furthermore since the network setup that we looked into was limited in the master-server setting it would be interesting to expand this type of accelerated algorithm to different type of networks, in a fashion similar to the work of Jakovetic [8], and let notions as consensus be a part of the algorithm and provide analysis upon that.

References

[1] Alekh Agarwal and John C. Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 873–881, 2011.

[2] Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167 – 175, 2003.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[4] Sébastien Bubeck. Theory of convex optimization for machine learning. *CoRR*, abs/1405.4980, 2014.

[5] I-An Chen. *Fast Distributed First-Order Methods*. PhD thesis, Massachusetts Institute of Technology, June 2012.

[6] Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. A delayed proximal gradient method with linear convergence rate. In *IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2014, Reims, France, September 21-24, 2014*, pages 1–6, 2014.

[7] Dusan Jakovetic, Joao Manuel Freitas Xavier, and Jose M. F. Moura. Convergence rates of distributed nesterov-like gradient methods on random networks. *Trans. Sig. Proc.*, 62(4):868–882, February 2014.

[8] Dusan Jakovetic, Joao Manuel Freitaso Manuel Freitas Xavier, and Jose M. F. M. F. Moura. Fast distributed gradient methods. *CoRR*, abs/1112.2972, 2011.

[9] Michael Athans John Tsitsiklis, Dimitri Bertsekas. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *Automatic Control, IEEE Transactions on (Volume:31 , Issue: 9)*, Sep 1986.

[10] Melanie L. Lenard and Michael Minkoff. Randomly generated test problems for positive definite quadratic programming. *ACM Trans. Math. Softw.*, 10(1):86–96, January 1984.

[11] Arkadi S. Nemirovsky and DB Yudin. Problem complexity and method efficiency in optimization. 1983.

[12] Yurii Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London, 2004.

[13] S. Sundhar Ram, Angelia Nedic, and Venugopal V. Veeravalli. Distributed stochastic subgradient projection algorithms for convex optimization. *J. Optimization Theory and Applications*, 147(3):516–545, 2010.

[14] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011.

[15] Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2510–2518. Curran Associates, Inc., 2014.

[16] Zeyuan Allen Zhu and Lorenzo Orecchia. A novel, simple interpretation of nesterov's accelerated method as a combination of gradient and mirror descent. *CoRR*, abs/1407.1537, 2014.