A Comparative Analysis of Graph Signal Recovery Methods for Big Data Networks

Alexandru Cristian Mara

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 06.10.2017

Thesis supervisor:

Prof. Alexander Jung

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Alexandru Cristian Mara

Title: A Comparative Analysis of Graph Signal Recovery Methods for Big Data Networks

Date: 06.10.2017 Language: English Number of pages: 6+45

Department of Computer Science

Professorship: Machine Learning And Data Mining (Macadamia)

Supervisor and advisor: Prof. Alexander Jung

Graph-based signal recovery (GSR) techniques have been successfully used in different domains for labelling complete graphs from partial subsets of given labels. Much research has been devoted to finding new efficient approaches for solving this learning problem. However, we have identified a lack of research in empirically comparing different GSR methods on big data graphs. In this work, we implement highly scalable versions of five state-of-the-art methods, which we benchmark under identical conditions on a number of real and synthetic datasets. We perform a comprehensive evaluation of these methods in terms of accuracy, scalability, robustness to noise and graph topology as well as sampling set selection. We find that recently proposed methods based on TV minimization outperform more classical approaches that measure the graphs smoothness through the quadratic form. We draw other interesting conclusions and discuss merits and faults of each of the methods studied.

Keywords: Graph signal processing, signal recovery, semi-supervised learning, transductive learning, GSR, GSSL, graphs, networks, big data, benchmark, review, label propagation

Preface

This thesis is submitted for the degree of Master of Science in Technology at the School of Science of Aalto University. The research presented herein has been conducted under the supervision of Prof. Alexander Jung within the *Machine Learning for Big Data group* of the Computer Science department. To the best of my knowledge, and with the only exception of appropriately indicated references to previous research, this work is original. Parts of this research have been published in the scientific paper referenced in the list of publications.

In the following lines, I would like to thank everyone who has helped or supported me during the long process of obtaining this degree. Firstly, I would like to express my deepest gratitude to my supervisor Alexander Jung who has been the best mentor I could have asked for. You introduced me to the field of graph signal processing, supported and guided me not only during the time I worked on this thesis but for the two years I have been part of your group. Your rigour, motivation and enthusiasm have inspired and encouraged me to pursue a career in research. It has been a real pleasure to be a part of your group.

I want to extend my gratitude to all the professors at the Computer Science department of Aalto University for their professionalism and the knowledge they have transmitted me. I also thank Prof. Alberto Mozo Velasco who introduced me to the field of machine learning and without whom I would have not been able to pursue an MSc in this area.

To my friends, colleagues and everyone I have met in Finland, thank you for making these some of the best years of my life and reminding me that work is not the only thing that matters. I will never forget you! Finally, I want to thank my family for their love, for believing in me and for their continuous support. And my dear Eli, thank you for always being there when I needed you.

Espoo, October 6, 2017

A. C. Mara

Contents

List of Publications

I A. Mara and A. Jung, *Recovery Conditions and Sampling Strategies for Network Lasso*, In proceedings of 51st Asilomar Conference on Signals, Systems and Computers, Forthcoming, 2017.

Symbols and abbreviations

Symbols

$\mathcal{Z}$	data points input space
$\mathcal{D}$	dataset
$\mathcal{G}$	graph
$\mathcal{V}$	set of graph vertices
$\mathcal{E}$	set of graph edges
$\mathbf{W}$	set of edge weights
$\mathcal{M}$	sample of nodes with known initial signal values
x_i	initially known signal of node $i \in \mathcal{M}$
$x[i]$	signal value associated to node $i \in \mathcal{V}$
$\hat{x}[i]$	predicted signal value for node $i \in \mathcal{V}$
$y[e]$	signal value associated to edge $e \in \mathcal{E}$
N	number of vertex signals (same as $\lvert\mathcal{V}\rvert$)
M	number of edge signals (same as $\lvert\mathcal{E}\rvert$)
$\mathcal{C}$	cluster within a graph
$\mathcal{F}$	partition of graph in set of clusters
$\partial\mathcal{F}$	the boundaries of a partition $\mathcal{F}$
ε	empirical error
$\mathcal{N}(i)$	neighbourhood of node i

Operators

$deg(x_i)$	degree of vertex i
$diag(\mathrm{x})$	matrix obtained by diagonalizing vector x
$\lVert\mathbf{D}\rVert_p$	p norm of a matrix
$\lVert\mathrm{x}\rVert_{\mathrm{TV}}$	total variation of graph signal

Abbreviations

API	application programming interface
CD	community detection
GSP	graph signal processing
GSR	graph signal recovery
GSSL	graph semi-supervised learning
HPC	high performance computing
LFR	Lancichinetti-Fortunato benchmark
LP	label propagation
NMSE	normalized mean square error
nLasso	network Lasso
SLP	sparse label propagation
SP	signal processing
SSL	semi-supervised learning

1 Introduction

The enormous variety of modern technological systems generate data at an unprecedented scale. This data is of different types (audio, video, text, ...) and constitutes the current big data scenario [16, 42]. A key asset in the modern global economy is not the raw data itself but the information that can be extracted by analysing it. One of the main steps in this data analysis process is labelling or classifying data. For example, determining objects of interest in images, filtering spam emails from interesting and non-interesting examples or identifying the 3D structure of proteins. The complexity of these tasks requires human experts with domain knowledge, making the process slow, expensive, prone to errors and in the case of huge amounts of information, impossible. For these reasons, the research in automated data analysis and labelling has gained popularity in recent years. One of the most active areas is semi-supervised learning or SSL [11], [47], which aims at labelling complete datasets by using the information provided by both labelled and unlabelled points.

In many applications, the data presents an inherent structure which can be used in the analysis and information extraction processes. The individual elements can be regarded as nodes and their interactions as edges of a data graph [6, 31]. This network or graph representation can reflect physical properties of the data, statistical dependencies or a combination of both. The recent field of graph signal processing (GSP) aims at analysing network structured data using concepts and tools from classical discrete signal processing. Some of the key problems studied within the field of GSP include dictionary learning [18], graph-based transforms [36], community detection (CD) [20], sampling of graph signals [1, 13] and graph signal recovery (GSR) [30]. In this study we focus on the problem of efficient graph signal recovery, which can be also interpreted as a specific type of semi-supervised learning from graphs (GSSL). The objective in GSR or GSSL is to infer the labels of all data points based on the graph structure and a small set of nodes with initially known labels. Within the framework of Markov random fields, a general classification problem similar to the GSR exists, called, the metric labelling problem [25]. The aim in this setting is to find a labelling for nodes of a given graph such that the joint cost of assigning labels to nodes (assignment cost) and cost of assigning different labels to connected nodes (separation cost) is minimized. Furthermore, signal recovery from graphs is also related to the uncapacitated assignment quadratic problem [19], widely studied in the field of discrete optimization. In this problem, a set of activities have to be matched with locations so that the sum of assignment costs of activities to locations and flow costs of activities that interact is minimized. We highlight the fact that GSR is a regression task rather than classification, thus, methods which require *a priori* knowledge of the number of different labels such as the ones presented in [5], [25] or [46], are not considered. The signal recovery problem is also often referred to as transductive learning, in contrast to inductive learning, which is not discussed here. The goal of inductive learning is to determine the appropriate labels for new

elements added to the data graph [44].

So far most of the research on GSR has been focused on different theoretical aspects such as analysing recovery strategies [12, 30] and sampling techniques [1, 13, 14]. Nevertheless, no research has been conducted what so ever to empirically analyse the accuracy and scalability of well known GSR algorithms on big data graphs. The closest to our work is [40], where the authors analyse and empirically compare three GSR methods applied to the specific problem of class-instance extraction. The real datasets used in this comparison are relatively small of only a few hundred thousand nodes. In [11] several SSL methods, including label propagation, are theoretically and empirically analysed. However, the datasets and graphs derived from them used in the analysis contained only a few thousand points. Authors proposing new methods, e.g. [21, 24, 48], usually provide limited and often insufficient experimental results comparing their methods against others. In addition, the algorithms are generally implemented on different platforms, thus, making them not comparable neither in terms of efficiency nor scalability. Moreover, the current big data scenario requires distributed and scalable platforms and algorithms. For instance, the SnapVX solver [22] provides a common framework for general convex optimization problems on graphs and can be used to implement many signal recovery algorithm. Nevertheless, this platform is not scalable nor flexible enough for our experiments (e.g. it is not designed to run on a cluster of machines and does not accept weighted graphs). For these reasons, we propose a benchmarking process using both real and synthetic graphs of varying topologies and sizes to compare state-of-the-art graph signal recovery methods implemented on the same big data framework. We, therefore, provide a common ground to conduct a fair comparison of different methods and determine the most salient ones.

Research hypothesis: Many studies confirm that GSR has become a main task in network science ([3, 39]). This problem has been addressed from various perspectives and through different algorithms and our main goal is to identify the most salient ones in terms of accuracy, scalability and convergence rate. Moreover, we assume that some methods, based on specific smoothness measures and consistency with initial labels, will perform better on certain graph topologies and graph signals characteristics than on others. For this reason, we intend to identify the optimal recovery methods for a variety of networks. At the same time, we seek to find how different network characteristics such as size, edge density, clustering coefficient, average degree, among others, impact the behaviour of the algorithms. For instance, we analyse to what extent the average degrees of graphs affect the algorithms execution times. Additionally, we study how other factors such as noise in the graph signals, weights that reflect or not the topology of graphs, sampling set selection strategies and sizes impact the signal recovery process.

1.1 Thesis Contributions

The two main contributions of this thesis are: (i) an empirical comparison of several state-of-the-art algorithms for GSR on both real and synthetic networks and (ii) we provide efficient distributed implementations of these algorithms for the big data

framework GraphX. Our results give guidance for selecting the most suitable method for a given recovery problem. Finally, we highlight the main advantages and disadvantages of each graph signal recovery approach studied.

1.2 Thesis Outline

The rest of this thesis is structured as follows. In Section 2 we summarize key concepts of graph theory, complex networks, signal recovery and distributed big data frameworks. Section 3 presents several GSR algorithms such as label propagation (LP), sparse label propagation (SLP) or network Lasso (nLasso), as well as their implementations on a big data framework. Section 4 presents the datasets used and benchmarking setup. In Section 5 we report our main results and finally, in Section 6 a summary of the thesis is presented as well as conclusions and future work.

2 Big Data over Networks

In this section, we first introduce some basic concepts related to graphs and complex networks. Then we focus on learning from graph structured data and, in particular, the graph signal recovery problem, which is the core concept of this thesis. Finally, we present the main characteristics of the GraphX big data framework.

2.1 Graph Theory

Graph theory is a branch of mathematics devoted to the study of graphs and their properties with applications in many areas such as chemistry, physics, biology, social sciences and computer science among others.

Graphs are mathematical structures which constitute a generic model for representing sets of objects, of any type, and their inter-dependencies. These objects or data points are encoded in the nodes of the graphs and their dependencies in the edges connecting the nodes. We can formalize these concepts as follows. For any dataset $\mathcal{D}$ containing N data points from a certain input space $\mathcal{Z}$, $\mathcal{D} = \{z_i\}_1^N \subseteq \mathcal{Z}$, we can define a simple undirected graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$. The graph contains a set of nodes or vertices $\mathcal{V} \in \mathbb{R}^N$. Each node $i \in \mathcal{V}$ corresponds to one of the data points z_i. The set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ contains the similarities between data points z_i and z_j as pairs $\{i, j\} \in \mathcal{E}$. The sparse weight matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ (symmetric for undirected graphs) contains the strengths of the connections. Each nonzero element $W_{i,j} > 0$ represents the strength of the connection $\{i, j\} \in \mathcal{E}$. If no edge between nodes i,j exists, i.e. $e = \{i, j\} \notin \mathcal{E}$, then $W_{i,j} = 0$.

Types of Graphs

Many different types of graphs can be identified based on their characteristics. Multigraphs present more than one single edge between a pair of nodes. Graphs with self-loops are those which have at least one node $i \in \mathcal{V}$ such that $\{i, i\} \in \mathcal{E}$. We refer by simple graphs to those with no self-loops and at most one edge connecting two nodes. Complete graphs present edges between every possible pair of nodes i.e. for all $i, j \in \mathcal{V}$ we have $\{i, j\} \in \mathcal{E}$. We distinguish between directed $\{i, j\} \in \mathcal{E} \neq \{j, i\} \in \mathcal{E}$ and undirected $\{i, j\} \in \mathcal{E} = \{j, i\} \in \mathcal{E}$ graphs. Weighted graphs are those whose edges carry an associated numerical value, $W_{i,j} > 0$, indicating the strength of the connection, while for unweighted graphs the edges either exist, $W_{i,j} = 1$, or not ,$W_{i,j} = 0$. Representations of each of this types of graphs are shown in Figure 1.

Graph Topology

Graph theory also focuses on the study of the topology of graphs and specific properties such as node degree, clustering coefficient, centrality measures (e.g. betweenness centrality), paths, subgraphs, components, connectedness and many others [4, 23]. In what follows we describe the most relevant measures for our study.

The node degree d_i of a vertex $i \in \mathcal{V}$ is defined as the number of neighbours $j \in \mathcal{N}(i)$ of i. We define the average network degree as: $d_{avg} = \sum_{i \in \mathcal{V}} d_i / N$ and the

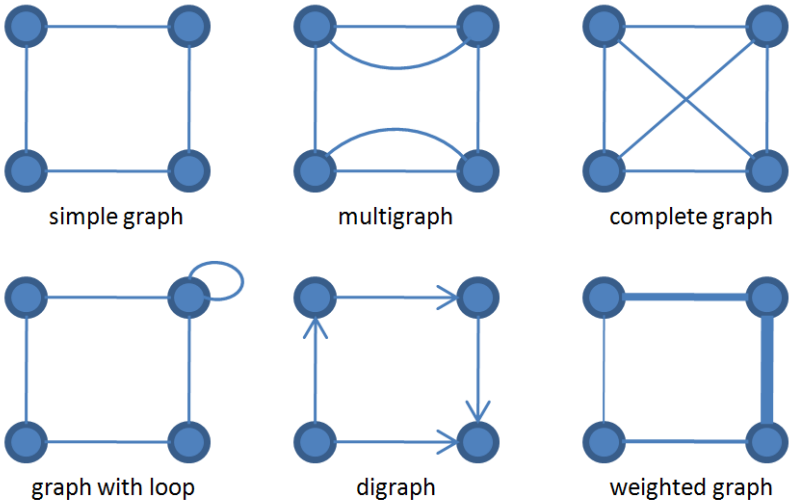

Figure 1: Basic types of graphs

degree distribution as the probability of a randomly selected node to have degree d, i.e. $P(d) = N_d/N$. The clustering coefficient C_i for a node $i \in \mathcal{V}$ is the number of edges between its neighbours out of all the possible ones. The average clustering coefficient for the whole network C_{avg} is, therefore $C_{avg} = \sum_{i \in \mathcal{V}} C_i/N$. Centrality measures indicate how important a node $i \in \mathcal{V}$ or edge $\{i, j\} \in \mathcal{E}$ is w.r.t. certain criteria. A path (or walk) is a sequence of vertices where consecutive pairs are connected by edges. For unweighted graphs, the path length is the number of traversed edges, while for weighted graphs is the sum of the inverse edge weights that belong to the path. A subgraph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*, \mathbf{W}^*)$ induced from a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ consists of a subset of nodes, $\mathcal{V}^* \subseteq \mathcal{V}$ and edges, $\mathcal{E}^* = \{\{i, j\} \in \mathcal{E} | i, j \in \mathcal{V}^*\}$. A component is a subgraph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*, \mathbf{W}^*)$ where a path exists between any pair of vertices $i, j \in \mathcal{V}^*$. If a graph $\mathcal{G}$ has a single component, it is said to be connected.

Graphs with more special structures can be found such as regular graphs, bipartite graphs and trees. Regular graphs have the same number d_i of neighbours per each node $i \in \mathcal{V}$. In bipartite graphs the set of vertices $\mathcal{V}$ can be separated into two disjoint sets $\mathcal{V} = S_1 \cup S_2$ so that the edges $\mathcal{E}$ connect one node from each set. Finally, trees are connected graphs with no loops.

Graph Representation

Graph structures can be represented in several ways. Two commonly used representations, in particular for storing graphs in computers, are edge lists and adjacency lists. Edge lists encode a graph $\mathcal{G}$ as a list of pairs of nodes (i, j), where i, j are nodes in $\mathcal{V}$ connected by an edge $\{i, j\} \in \mathcal{E}$. An adjacency list, on the other side, represents a graph $\mathcal{G}$ as a list of neighbours for each node $i \in \mathcal{V}$. Neither of this representations is, in fact, useful for mathematical computations, so other structures are required.

An adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ stores the same information as the adjacency

list, but in a much more convenient matrix form. Formally we define $\mathbf{A}$ as:

$$A_{i,j} = \begin{cases} 1 & \text{if } \{i,j\} \in \mathcal{E} \\ 0 & \text{else} \end{cases} \tag{1}$$

If the graph $\mathcal{G}$ represented by adjacency matrix $\mathbf{A}$ is weighted, then $\mathbf{A} = \mathbf{W}$.

Graphs can also be represented in matrix form using the discrete Laplacian, or Laplacian matrix $\mathbf{L} \in \mathbb{R}^{N \times N}$. The subfield of graph theory which represents graphs as matrices and studies their properties is known as spectral graph theory [8, 15]. In particular, spectral graph theory studies the eigenvalues and eigenvectors of the graph Laplacian. The (unnormalized) Laplacian matrix $\mathbf{L}$ is defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{2}$$

Where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix. Equivalently, we can define $\mathbf{L}$ element-wise as:

$$L_{i,j} = \begin{cases} deg\{i\} & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } \{i,j\} \in \mathcal{E} \\ 0 & \text{else} \end{cases} \tag{3}$$

The last representation of a graph we discuss here is the incidence matrix. For a directed graph $\overrightarrow{\mathcal{G}}$, we define the incidence matrix $\mathbf{B} \in \mathbb{R}^{M \times N}$ [38] as:

$$B_{e,i} = \begin{cases} \mathbf{W}_e & \text{if } i = e^+ \\ -\mathbf{W}_e & \text{if } i = e^- \\ 0 & \text{else.} \end{cases} \tag{4}$$

If a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ is undirected, we can induce an orientation by declaring for each edge $e = \{i,j\} \in \mathcal{E}$ one node as the head e^+ (e.g., $e^+ = \{i\}$) and the other node as the tail e^- (e.g., $e^- = \{j\}$). We then obtain an oriented graph $\overrightarrow{\mathcal{G}} = (\mathcal{V}, \overrightarrow{\mathcal{E}}, \mathbf{W})$ from which the incidence matrix $\mathbf{B}$ can be computed.

2.2 Complex Networks

While there is no single widely accepted definition of when a graph is a complex network; in the fields of social sciences and computer science, the name complex network is used to refer to the study of graphs which do not have a regular structure, such as grid or chain graphs. These simple non-bipartite graphs emerge from natural phenomena and present specific topological features such as: average clustering coefficients higher than those of random networks of the same size (i.e. higher than $C_{avg} = d_{avg}/N$), nodes organized in hierarchical structures and power law distribution of node degrees (very few nodes with high degree or hubs and most nodes with relatively low degree). In addition, these networks present a short average path length between any pair of nodes (small world property [41]). Therefore, starting from a random node, any other can be reached in very few hops, usually in the order

of $log(N)$. Moreover, due to the power law distribution of node degrees ($P(d) \sim d^{-\gamma}$ where γ is typically in the range $2 < \gamma < 3$) and the existence of hubs, complex networks are often referred to as scale free networks ([2]). Also, as mentioned, complex networks very often can be partitioned in a set of communities or clusters $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2 \dots\}$ so that each element $i \in \mathcal{C}_l$ is strongly connected to other nodes $j \in \mathcal{C}_l$ but weakly connected to nodes $k \in \mathcal{V} \backslash \mathcal{C}_l$.

In order to evaluate the performance of graph processing algorithms, many models for generating networks with similar properties to the above-mentioned have been proposed. In Section 4, we use one of this models, particularly the LFR model, to generate synthetic test graphs. The LFR model proposed by Lancichinetti and Fortunato [26] is a generalization of the popular GN model. The GN model by Girvan et. al. [20] is a simple method to create community structured graphs where nodes have very similar degrees, communities present roughly the same topology and sizes and are separated from each other in the same manner. The LFR model extends the GN to accommodate power law distributions of node degrees and community sizes.

2.3 Learning from Big Data over Networks

Representing data as graphs or networks has many advantages, such as providing a general computation model where algorithms do not have to be tailored to each problem, efficient and scalable graph processing methods are available, mostly any kind of data can be represented by graphs and the structural features of the data can make traditional data mining approaches infeasible. The process of identifying and extracting useful information from network structured data is called graph mining. This process has applications in many fields such as detecting communities of users in social media channels, identifying anomalies in computer networks, studying particles for structure analysis or finding frequent molecular structures in biology.

Graph Signals

In many applications, labels or signals containing additional information can be associated to the nodes and edges of graphs. These labels can be numerical or categorical although they are usually converted to numerical values to simplify processing. We represent node signals over a graph $\mathcal{G}$ by a vector $x[\cdot] \in \mathbb{R}^N$ which assigns a value $x[i] \in \mathbb{R}$ to each node $i \in \mathcal{V}$ i.e:

$$x[\cdot] = [x[1], x[2] \dots x[N]]^T \in \mathbb{R}^N \tag{5}$$

We represent edge signals as a vector $y[\cdot] \in \mathbb{R}^M$ which assigns a value $y[i] \in \mathbb{R}$ to each edge $e \in \mathcal{E}$ i.e:

$$y[\cdot] = [y[1], y[2] \dots y[M]]^T \in \mathbb{R}^M \tag{6}$$

Examples of signals can be user ratings on a graph whose nodes represent purchased products, elevation information for a graph of intersections connected by roads, political views of users in a social network or company sizes in financial graphs.

Graph Signal Recovery

In order to analyse graph signals, the recent field of graph signal processing [37, 39] explores the extension and adaptation of tools and methods from classical discrete signal processing to graph structured data. An example of such an extension is the graph Fourier transform [36]. Since obtaining the real labels or signal values x_i for all $i \in \mathcal{V}$ is an expensive process, generally, only a small subset of the graph nodes are labelled i.e. $x_i \in \mathcal{M}$ with $|\mathcal{M}| \ll N$. We represent the initially available labelled nodes as a set $\mathcal{M} \subseteq \mathcal{V}$.

Graph signal recovery techniques aim at inferring the signal values $x[i]$ for all $i \in \mathcal{V}$ based on the available ones $x_i \ \forall i \in \mathcal{M}$ and the graph topology. Different methods measure the empirical error, $E(\hat{x}[\cdot])$, between the initially known x_i and recovered signals $\hat{x}[i]$ via different norms (e.g. ℓ_1-norm or ℓ_2-norm) or impose certain constraints, e.g. $E(\hat{x}[\cdot]) = 0$. In addition to this, the labels are required to be consistent with the geometry induced by the graph, i.e. follow the smoothness assumption of SSL. The smoothness assumption, also known as clustering or manifold assumption [11], states that close by points i, j should have similar signal values $x[i], x[j]$. This, in turn, results in the decision boundaries lying in low density regions. Different smoothness measures $\mathcal{R}$(x) can be defined, which results in different recovery problems.

The problem of signal recovery from graph structured data, which will be analysed in detail in the next sections, can be also formulated as a convex optimization problem. In this problem smoothness of the graph signal is enforced but, at the same time, a small discrepancy of the predicted signals to the nodes in the sampling set $\mathcal{M}$ is encouraged, i.e.:

$$\hat{x} \in \underset{\hat{x}[\cdot] \in \mathbb{R}^{\mathcal{V}}}{\arg\min}\, E(\hat{x}[\cdot]) + \lambda \mathcal{R}(\hat{x}[\cdot]) \tag{7}$$

The parameter λ allows to trade between empirical error and smoothness of the recovered signal.

2.4 Big Data framework

Since it was released by Google in 2004, the MapReduce programming model [17] has become the *de facto* approach for big data processing on clusters of machines. It consists of a distributed phase or *map*, where the data is processed in small independent chunks on each cluster node and an information gathering phase or *reduce*, where the output of the maps is tree-aggregated. This simple architecture, together with a powerful run-time system able to automatically partition data, schedule execution and provide fault tolerance, revolutionized the data analytics field. Apache Hadoop was the first open source implementation of the MapReduce paradigm. Many companies adopted the architecture due to its usability and simple administration. The platform does not require specialized hardware but rather a cluster of commodity machines while providing orders of magnitude faster processing compared to traditional HPC architectures.

The Apache Spark project [45], started by researchers at UC Berkeley in early 2009, is the natural evolution of Hadoop. In 2016, Apache Hadoop was replaced

by Spark as the top open-source big data project. This data flow management framework extends the basic *map* and *reduce* primitives with new flexible operations designed for in-memory iterative computing. Unlike Hadoop, Spark stores intermediate results in the cluster RAM memory, thus, becoming the natural choice for multi-pass algorithms. Spark also uses a similar Master/Slave model as Hadoop. In Spark a cluster is composed of one or more machines each with one or more CPU threads. Generally, each thread is one node of the cluster with its own computation capabilities and private memory space. One of the cluster nodes is the master and controls data partitioning, execution scheduling and fail recovery. The remaining nodes or workers, process their local information (assigned data partitions) and communicate the outputs to the next designated workers.

GraphX platform

The Spark core is surrounded by a huge ecosystem of specialized processing tools. One such tool is GraphX [43], which adds efficient and scalable graph processing capabilities to Spark. This platform is one of the most recent and advanced tools for distributed graph processing available. Some of its main characteristics are the capacity to scale to graphs of any size, a flexible API, simplicity of use and management, a variety of graph algorithms available and a broad and active community of developers. Moreover, the platform can be used from popular programming languages such as Python, Scala and Java. For these reasons, GraphX is the platform selected to implement our algorithms and conduct the experiments of this work.

In order to accommodate data in a graph structure, GraphX like most similar systems models the information as a property graph [34]. Property graphs are directed multigraphs which contain user-defined properties (labels) associated with vertices and edges. The system exposes methods to manipulate these properties, as well as the graph structure itself, making graph computation possible.

Providing an in depth description of the architecture of GraphX and its internal optimizations are beyond the scope of this work. Nevertheless, it is worth mentioning that the property graphs are distributed across cluster nodes using a vertex-cut approach. This optimized parallelization strategy resumes to splitting graphs along edges, rather than vertices, as shown in Figure 2. Therefore, the vertex-cut strategy assigns sets of edges to individual cluster nodes and allows vertices to span multiple nodes. Internally, graphs are represented as a pair of collections of vertices and edges where each can be hash-partitioned according to a different strategy. A routing table encodes the edge partitions for each vertex. Figure 3, presented as Image 3 in [43], depicts the distributed graph representation in GraphX.

As already mentioned, the Spark platform, including GraphX, is designed for cluster computing. This type of computing requires a cluster manager, a software aware of the available hardware resources. Among other tasks the cluster manager is used to deploy the cluster, inform the master node about the availability of workers and their memory, synchronize the machines in the cluster and recover crashed machines. In Spark three managers can be used: standalone, mesos or yarn.

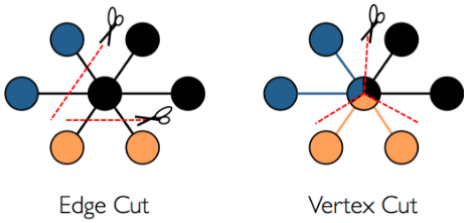

Figure 2: Representations of vertex and edge cuts. Image from the GraphX programming guide.

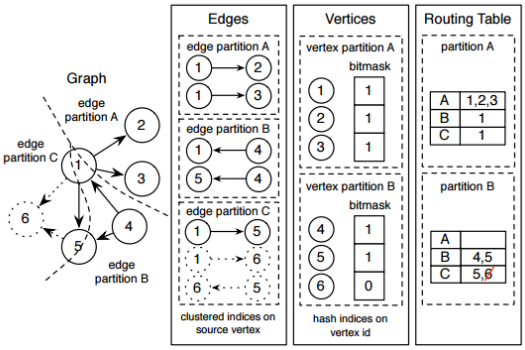

Figure 3: Distributed representation of a simple graph in GraphX. The edge, vertex and routing table partitions do not have direct correspondence to cluster nodes as this depends on the cluster manager and on their particular partitioning strategies. Vertex 6 and his incident edges have been restricted from the graph so are represented with dotted lines.

3 Learning Algorithms

In this section we detail the algorithm for signal recovery over graphs benchmarked. We present, for each method, its associated optimization problem and distributed implementation in GraphX. The selected algorithms are a community detection-based labelling, label propagation, sparse label propagation and network Lasso. Additionally, we include an average consensus method to be used as baselines for the recovery problem.

As mentioned in Section 2, most of these methods can be cast into a common framework where the aim is to obtain a signal that minimizes Eq. 7. For convenience, this equation is again presented below:

$$\hat{x} \in \underset{\hat{x}[\cdot]\in\mathbb{R}^{\mathcal{V}}}{\arg\min}\, E(\hat{x}[\cdot]) + \lambda\mathcal{R}(\hat{x}[\cdot])$$

The algorithms we detail next measure the smoothness of the graph signal ($\mathcal{R}(\hat{x}[\cdot])$) and empirical error ($E(\hat{x}[\cdot])$) with the initial labels via different norms or impose certain constraints on them.

3.1 Average Consensus

The simplest method to label a complete graph $\mathcal{G}$ is to select a signal value $\hat{x}[i]$ constant for all nodes $i \in \mathcal{V}$. This method, to which we will refer as *Avg_cons* in what follows, does not take into consideration the graph topology nor edge weights (i.e. does not minimize any smoothness measure $\mathcal{R}(\hat{x}[\cdot])$) and also does not require any parameters. The constant signal value $\hat{x}[i]$, is obtained as a consensus between the nodes of the graph with *a priori* known labels $x_i \in \mathcal{M}$. Therefore, this method amounts to simply minimize the empirical error w.r.t. the initial labels i.e:

$$\hat{x} \in \underset{\hat{x}[\cdot]\in\mathbb{R}^{\mathcal{V}}}{\arg\min}\, E(\hat{x}[\cdot]) \tag{8}$$

Here, the empirical error is defined as the sum of the squared differences between the predicted signal value $\hat{x}[i]$ at each node $i \in \mathcal{M}$ and the initially known label x_i, this is:

$$E(\hat{x}[\cdot]) = \sum_{i\in\mathcal{M}} ||\hat{x}[i] - x_i||_2^2 \tag{9}$$

A closed form solution for solving the minimization problem can be derived. The resulting consensus signal value is then:

$$\hat{x}[\cdot] = \frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}} x_i \tag{10}$$

Distributed implementation

The average consensus algorithm can be implemented in GraphX in a simple non-iterative fashion. The closed form solution to obtain the consensus signal $\hat{x}[\cdot]$,

can be computed using the atomic operations *reduce* and *broadcast*. The *reduce* operation computes the average signal of elements $i \in \mathcal{M}$. The *broadcast* sets the values $x[i]$ for all i $\in \mathcal{V}$ to the computed average value. The distributed formulation is presented in Algorithm 1.

Algorithm 1 Distributed Avg_cons algorithm

Input: sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i\in\mathcal{M}}$.

1: $\tilde{x} = 1/|\mathcal{M}| \sum_{i\in\mathcal{M}} x_i$

2: for all nodes $i \in \mathcal{V}$: $\hat{x}[i] = \tilde{x}$

Output: Labelled points $\hat{x}[i]$ $\forall i \in \mathcal{V}$

3.2 Community-based labelling

Another simple method to recover the signal values of the nodes $i \in \mathcal{V}$ of a graph $\mathcal{G}$ is through a community detection algorithm. We assume a graph $\mathcal{G}$ can be partitioned into a number of clusters $\mathcal{F} = \{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$. This partitioning is found by applying the LP for community detection algorithm [33]. According to the smoothness assumption of semi-supervised learning, it is reasonable to consider that nodes from the same cluster should share the same signal values. Therefore, for each cluster $\mathcal{C}_l \in \mathcal{F}$ a consensus signal value $\hat{x}_l^{cons}$ is selected for all nodes $i \in \mathcal{C}_l$ as the most frequent x_i of all $i \in \mathcal{C}_l$, i.e.:

$$\hat{x}_l^{cons} = \text{ mode}(x_i) \text{ for all } i \in \mathcal{C}_l \cap \mathcal{M} \tag{11}$$

The community detection algorithm uses the graph topology to determine the clusters, although, it does not make use of the edge weights nor minimizes any smoothness measure for the graph signal. In order to identify the community structure, the algorithm starts by giving each node $i \in \mathcal{V}$ its own cluster. Then the clusters are iteratively joined by making each node adopt the most frequent cluster label of its neighbours, i.e. $\hat{x}_l^{cons}$. The community detection algorithm, with the additional step to recover graph signals, which we will refer to as *LP_cd*, is presented below as Algorithm 2.

Distributed implementation

The community-based labelling method first, uses the LP for community detection algorithm, implemented in GraphX, to find the partitioning $\mathcal{F}$ of the graph $\mathcal{G}$ in a set of clusters. Then, for each cluster $\mathcal{C}_l$, the mode of the signal values of elements $i \in \mathcal{C}_l \cap \mathcal{M}$ is computed using a *reduce* operation. This value is spread to all vertices in the same cluster via a *join* operation. The pseudocode of the distributed LP_cd method is presented as Algorithm 3.

Algorithm 2 LP_cd algorithm

Input: data graph $\mathcal{G}$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i\in\mathcal{M}}$.
Initialize: $k = 1$, for each node i, $C_i^{(0)} = i$.

1: **repeat**
2: Set an order to select nodes in the graph
3: For each node i chosen in that specific order, $C_i^k = \text{mode}(C_j^{k-1})$ for $j \in \mathcal{N}(i)$
4: k = k + 1
5: **until** stopping criteria is satisfied
6: **for** $\mathcal{C}_l \in \mathcal{F}$ **do**
7: $\hat{x}_l^{cons} =$ mode(x_i) for all $i \in \mathcal{C}_l \cap \mathcal{M}$
8: **end for**

Output: Labelled points $\hat{x}[i]$ $\forall i \in \mathcal{V}$

Algorithm 3 Distributed LP_cd algorithm

Input: data graph $\mathcal{G}$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i\in\mathcal{M}}$.
Initialize: $k = 1$, for each node i, $C_i^{(0)} = i$.

1: Obtain partitioning $\mathcal{F}$ via LPA
2: **for** $\mathcal{C}_l \in \mathcal{F}$ **do**
3: for all nodes $i \in \mathcal{C}_l \cap \mathcal{M}$: $\hat{x}_l^{cons} =$ mode(x_i)
4: for all nodes $i \in \mathcal{C}_l$: $\hat{x}[i] = \hat{x}_l^{cons}$
5: **end for**

Output: Labelled points $\hat{x}[i]$ $\forall i \in \mathcal{V}$

3.3 Label Propagation

The label propagation algorithm proposed by Zhu et. al. in 2002 [48], is one of the first methods for signal recovery from graphs. The algorithm uses the graph structure to propagate known labels of nodes to their unlabelled neighbours. In label propagation, nodes set their local signal value to the weighted average of their neighbours (Step 2 of Algorithm 4). As a consequence of this propagation strategy, the algorithm tends to smooth the graph signals or, in other words, penalize rapid changes between signals of connected nodes. This behaviour can be also seen as an optimization problem where the smoothness of the graph signal $\mathcal{R}(\hat{x}[\cdot])$ is measured through the quadratic form:

$$\mathcal{R}(\hat{x}[\cdot]) = \sum_{\{i,j\}\in\mathcal{E}} W_{i,j}||\hat{x}[i] - \hat{x}[j]||_2^2 \tag{12}$$

At the same time, the empirical error is forced to be $E(\hat{x}[\cdot]) = 0$ by clamping the signal values of nodes $i \in \mathcal{M}$ to their initial values after each iteration (Step 3 of

Algorithm 4). The complete LP algorithms, thus, minimizes:

$$\hat{x} \in \arg\min_{\hat{x}[\cdot]\in\mathbb{R}^{\mathcal{V}}} \sum_{\{i,j\}\in\mathcal{E}} W_{i,j}||\hat{x}[i]-\hat{x}[j]||_2^2 \text{ s.t. } \hat{x}[i]=x_i \text{ for all } i\in\mathcal{M} \tag{13}$$

Equivalently we can write the previous equation in terms of the graph Laplacian:

$$\hat{x} \in \arg\min_{\hat{x}[\cdot]\in\mathbb{R}^{\mathcal{V}}} \hat{\mathrm{x}}^T L\hat{\mathrm{x}} \text{ s.t. } \hat{x}[i]=x_i \text{ for all } i\in\mathcal{M} \tag{14}$$

The pseudocode of LP is presented as Algorithm 4. We will refer to this algorithm as *LP_sr* in order to distinguish it from the LP for community detection presented earlier.

Algorithm 4 Label Propagation (Zhu and Ghahramani, 2002)

Input: weight matrix $\mathbf{W}\in\mathbb{R}^{\mathcal{V}\times\mathcal{V}}$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i\in\mathcal{M}}$.
Initialize: $k=0$, $D_{i,i}\leftarrow\sum_j W_{i,j}$, $\hat{x}^0[\cdot]=(x_1,\ldots,x_{\mathcal{M}},0,0,\ldots,0)$
1: **repeat**
2: $\hat{x}^{k+1}[\cdot]=D^{-1}W\hat{x}^k[\cdot]$
3: $\hat{x}^{k+1}[i]=x_i$ for all nodes $i\in\mathcal{M}$
4: k = k + 1
5: **until** until convergence criteria is satisfied
Output: Labelled points $\hat{x}^k[i]\ \forall i\in\mathcal{V}$

Algorithm 5 Distributed Label Propagation

Input: weight matrix $\mathbf{W}\in\mathbb{R}^{\mathcal{V}\times\mathcal{V}}$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i\in\mathcal{M}}$.
Initialize: $k=0$, $\hat{x}^0[\cdot]=(x_1,\ldots,x_{\mathcal{M}},0,0,\ldots,0)$
1: **repeat**
2: for all nodes $i\in\mathcal{V}$: $\hat{x}^{(k+1)}[i]=\sum_{j\in\mathcal{N}(i)} W_{i,j}\hat{x}^{(k)}[j]/\sum_{j\in\mathcal{N}(i)} W_{i,j}$
3: for all nodes $i\in\mathcal{M}$: $\hat{x}^{(k+1)}[i]=x_i$
4: k = k + 1
5: **until** until convergence criteria is satisfied
Output: Labelled points $\hat{x}^k[i]\ \forall i\in\mathcal{V}$

Distributed implementation

The LP algorithm is implemented using the power iteration method. This technique consists of iteratively setting the signal $\hat{x}[i]$ of each node $i\in\mathcal{V}$ to the weighted average of its neighbours via *collect neighbours* and *map* operations. Then, using a *join* operation, the nodes in the sampling set are reset to their initially known values. This distributed implementation is shown as Algorithm 5.

3.4 Sparse Label Propagation

Sparse label propagation (SLP) is a recently proposed method by Jung et. al. [24]. It regards GSR as a non-smooth convex optimization problem. This problem is then solved by adapting the Pock-Chambolle primal-dual method [10], initially proposed for images, to graphs of arbitrary topologies. The algorithm also uses diagonal preconditioning [32] in order to improve convergence.

In simple terms, SLP can be seen as: for each node $i \in \mathcal{V}$ its incident edge signal values, $\{\hat{y}[\{i,j\}]\}_{j \in \mathcal{N}(i)}$ are first computed as the weighted difference between $x[i]$ and each $x[j]$ for all $j \in \mathcal{N}(i)$. In a second step, the signal value of each node is set to the difference between its current predicted value $\hat{x}[i]$ and the average of its incident edge values. A parameter λ allows leveraging the weights of the updates on the primal and dual variables. By increasing the value of λ more weight can be given to the update of the dual variable (the vertex signal value as an average of the incident edge values), and less on the primal (cumulative edge value updates) making the method similar to plain label propagation.

In SLP the smoothness of the graph signal $\hat{x}[\cdot] \in \mathbb{R}^{\mathcal{V}}$ is quantified via its total variation (TV) defined as:

$$\mathcal{R}(\hat{x}[\cdot]) = \sum_{\{i,j\} \in \mathcal{E}} W_{i,j} ||\hat{x}[i] - \hat{x}[j]||_1 \tag{15}$$

The empirical error between the predicted and known signal values of vertices $i \in \mathcal{M}$ is defined as $E(\hat{x}[\cdot]) = 0$. Therefore, the resulting minimization problem is:

$$\hat{x} \in \arg\min_{\hat{x}[\cdot] \in \mathbb{R}^{\mathcal{V}}} \sum_{\{i,j\} \in \mathcal{E}} W_{i,j} ||\hat{x}[i] - \hat{x}[j]||_1 \text{ s.t. } \hat{x}[i] = x_i \text{ for all } i \in \mathcal{M} \tag{16}$$

It can be observed that the problem solved by LP and SLP are very similar. While LP amount to minimizing the squared ℓ_2-norm, SLP aims to minimize the weighted ℓ_1-norm between the end nodes signals $\hat{x}[i]$ and $\hat{x}[j]$ of each edge $\{i,j\} \in \mathcal{E}$. The use of these different norms should result in LP better learning smooth graph signals, while SLP should be able to learn signals which exhibit abrupt value changes.

The pseudocode of SLP is presented below as Algorithm 6.

Distributed implementation

For SLP, the simple structures of the objective function and constraint set, when considered independently from each other, suggest the use of efficient proximal methods. In particular, as already mentioned, a preconditioned variant of the primal-dual method by Pock and Chambolle is used. Through operations such as *mapVertices* and *mapEdges* the signals associated with nodes and edges are updated iteratively. The nodes $i \in \mathcal{M}$ are clamped to their original values using a *join* operation. The GraphX implementation of this method is presented in Algorithm 7.

Algorithm 6 Sparse Label Propagation

Input: oriented data graph $\overrightarrow{\mathcal{G}}$ with incidence matrix $\mathbf{B} \in \mathbb{R}^{\overrightarrow{\mathcal{E}} \times \mathcal{V}}$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i \in \mathcal{M}}$.

Initialize: $k = 0$, $\hat{x}^0[\cdot] = (x_1, \ldots, x_{\mathcal{M}}, 0, 0, \ldots, 0)$, $\hat{y}^0[\cdot] = (0, \ldots, 0)$, $\gamma_i = 1/\lambda \sum_{j \in \mathcal{N}(i)} W_{i,j}$, $\psi_{\{i,j\}} = \lambda/(2W_{i,j})$.

1: **repeat**

2: $\hat{x}^{k+1}[\cdot] = \hat{x}^k[\cdot] - \mathbf{\Gamma B}^T \hat{y}^k[\cdot]$ with $\mathbf{\Gamma} = \mathrm{diag}\{\gamma_i\}_{i \in \mathcal{V}}$

3: $\hat{x}^{k+1}[i] = x_i$ for all sampled nodes $i \in \mathcal{M}$

4: $\tilde{x}[\cdot] = 2\hat{x}^{k+1}[\cdot] - \hat{x}^k[\cdot]$

5: $\hat{y}^{k+1}[\cdot] = \hat{y}^k[\cdot] + \mathbf{\Psi B}\tilde{x}[\cdot]$ with $\mathbf{\Psi} = \mathrm{diag}\{\psi_{\{i,j\}}\}_{\{i,j\} \in \mathcal{E}}$

6: $y^{k+1}[e] = y^k[e] / \max\{1, |y^k[e]|\}$ for all edges $e \in \overrightarrow{\mathcal{E}}$

7: $k = k + 1$

8: **until** stopping criterion is satisfied

Output: labels $\hat{x}^k[i]$ for all $i \in \mathcal{V}$

Algorithm 7 Distributed SLP algorithm

Input: oriented data graph $\overrightarrow{\mathcal{G}} = (\mathcal{V}, \overrightarrow{\mathcal{E}}, \mathbf{W})$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i \in \mathcal{M}}$.

Initialize: $k = 0$, $\mathbf{z}^{(0)} = \mathbf{0}$, $\mathrm{x}^{(0)} = \mathrm{x}_{\mathcal{M}}$, $\hat{\mathrm{x}}^{(0)} = 0$, $\hat{\mathrm{y}}^{(0)} = \mathbf{0}$, $\gamma_i = 1/\lambda \sum_{j \in \mathcal{N}(i)} W_{i,j}$, $\psi_{\{i,j\}} = \lambda/(2W_{i,j})$.

1: **repeat**

2: for all nodes $i \in \mathcal{V}$: $\tilde{x}^{(k+1)}[i] = \tilde{x}^{(k)}[i] - \gamma_i \Big[\sum_{j \in \mathcal{N}_+(i)} W_{i,j} \hat{y}^{(k)}[\{i,j\}] - \sum_{j \in \mathcal{N}_-(i)} W_{i,j} \hat{y}^{(k)}[\{i,j\}] \Big]$

3: for all nodes $i \in \mathcal{M}$: $\hat{x}^{(k+1)}[i] = x_i$

4: for all nodes $i \in \mathcal{V}$: $\tilde{x}[i] = 2\hat{x}^{(k+1)}[i] - \hat{x}^{(k)}[i]$

5: for all edges $e \in \overrightarrow{\mathcal{E}}$: $\hat{y}^{(k+1)}[e] = \hat{y}^{(k)}[e] + \psi_{\{i,j\}}(\tilde{x}[e^+] - \tilde{x}[e^-])$

6: for all edges $e \in \overrightarrow{\mathcal{E}}$: $\hat{y}^{(k+1)}[e] = \hat{y}^{(k+1)}[e] / \max\{1, |\hat{y}^{(k+1)}[e]|\}$

7: $k = k + 1$

8: **until** stopping criterion is satisfied

Output: labels $\hat{x}^{(k)}[i]$ for all $i \in \mathcal{V}$

3.5 Network Lasso

Network Lasso is a general framework to solve many optimization problems proposed in 2015 by Hallac et. al. [21]. The authors specialize the Lasso estimator, used in statistical learning with sparsity constraints, for graph structured data. A highly scalable implementation is provided based on the alternating direction method of multipliers or ADMM. Moreover, the authors show that many optimization problems can be cast into this framework.

In a recent paper by A. Mara and A.Jung [**I**], the Network Lasso or nLasso is specialized for the GSR problem and conditions for this recovery to be accurate are derived. We will now present this variation, the nLasso algorithm.

Like SLP, nLasso uses the TV as a measure of signal smoothness, i.e. $\mathcal{R}(\hat{x}[\cdot]) = \|\hat{\mathrm{x}}[\cdot]\|_{\mathrm{TV}}$. Nevertheless, unlike SLP, nLasso allows the initial values of sampled nodes $i \in \mathcal{M}$ to vary. The rationale behind this idea is that, if the initial samples contain measurements errors, by allowing their signals to change, the recovered graph signal might resemble more the original signal. The deviation of the predicted signal values from the original ones or empirical error ($E(\hat{x}[\cdot])$) is computed using the ℓ_1-norm, i.e:

$$E(\hat{x}[\cdot]) = \sum_{i \in \mathcal{M}} ||\hat{x}[i] - x_i||_1$$

In order to recover a graph signal with small TV and small empirical error, a natural strategy is to solve the minimization problem (17), where parameter λ allows to trade between signal smoothness and empirical error.

$$\hat{x} \in \operatorname*{arg\,min}_{\hat{x}[\cdot] \in \mathbb{R}^{\mathcal{V}}} \sum_{i \in \mathcal{M}} ||\hat{x}[i] - x_i||_1 + \lambda \sum_{\{i,j\} \in \mathcal{E}} W_{i,j} ||\hat{x}[i] - \hat{x}[j]||_2 \tag{17}$$

The ADMM updates of the nLasso algorithm, which are very similar to those in [21], are presented in Algorithm 8. The values u_{ij} are the scaled dual variables and $\rho > 0$ is a penalty parameter.

Distributed implementation

The general ADMM steps for nLasso have already been presented in Alg. 8. However, here we include the updates of $\hat{x}[i]^{k+1}$ for all $i \in \mathcal{V}$ and show the complete distributed algorithm. For the sampled nodes $i \in \mathcal{M}$, the signal update $\hat{x}[i]^{k+1}$ reads:

$$\hat{x}[i]^{k+1} = y[i] + S_{1/(\rho|\mathcal{N}(i)|)} \left(\frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} (z_{ij}^k - u_{ij}^k) - y[i] \right), \tag{18}$$

Here S is the soft thresholding operator defined as $S = sign(x) \cdot max(|x| - \lambda, 0)$ (cf. [7]). The updates for $\hat{x}[i]^{k+1}$ of nodes $i \notin \mathcal{M}$ is specialized by:

$$x[i]^{k+1} = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} (z_{ij}^k - u_{ij}^k). \tag{19}$$

The signal values $\hat{x}[i]$ associated with each node are updated through *collectEdges* and *join* operations, while the values z and u are computed for each edges through

Algorithm 8 ADMM updates for nLasso

Input: data graph $\mathcal{G}$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i\in\mathcal{M}}$.
Initialize: $k=0$, $\hat{x}^0[\cdot]=(x_1,...,x_{\mathcal{M}},0,0,...,0)$, $z^0[\cdot]=\hat{x}^0[\cdot]$, $u^0[\cdot]=(0,\ldots,0)$.

1: **repeat**
2: $\hat{x}[i]^{k+1}=\arg\min_{\hat{x}[\cdot]\in\mathbb{R}^{\mathcal{V}}}\left(||\hat{x}[i]^k-x_i||_1+\sum_{j\in\mathcal{N}(i)}(\rho/2)||\hat{x}[i]^k-z^k_{i,j}+u^k_{i,j}||_2^2\right)$
3: $\theta=\max\left(0.5,1-\frac{\lambda W_{ij}}{\rho|x[i]^{k+1}+u^k_{ij}-x[j]^{k+1}-u^k_{ji}|}\right)$
4: $z^{k+1}_{ij}=\theta(x[i]^{k+1}+u^k_{ij})+(1-\theta)(x[j]^{k+1}+u^k_{ji})$
5: $z^{k+1}_{ji}=\theta(x[j]^{k+1}+u^k_{ji})+(1-\theta)(x[i]^{k+1}+u^k_{ij})$
6: $u^{k+1}_{ij}=u^k_{ij}+(x[i]^{k+1}-z^{k+1}_{ij})$
7: $u^{k+1}_{ji}=u^k_{ji}+(x[j]^{k+1}-z^{k+1}_{ji})$
8: $k=k+1$
9: **until** stopping criterion is satisfied

Output: labels $\hat{x}^k[i]$ for all $i\in\mathcal{V}$

mapEdges. The pseudocode of the distributed algorithms implemented in GraphX is shown as Algorithm 9.

Algorithm 9 ADMM updates for nLasso

Input: oriented data graph $\overrightarrow{\mathcal{G}}$ with incidence matrix $\mathbf{B} \in \mathbb{R}^{\overrightarrow{\mathcal{E}} \times \mathcal{V}}$, sampling set $\mathcal{M}$, initial labels $\{x_i\}_{i \in \mathcal{M}}$.

Initialize: $k = 0$, $\hat{x}^0[\cdot] = (x_1, ..., x_{\mathcal{M}}, 0, 0, ..., 0)$, $z^0[\cdot] = \hat{x}^0[\cdot]$, $u^0[\cdot] = (0, \ldots, 0)$.

repeat
  for $i \in \mathcal{V}$ **do**
    if $i \in \mathcal{M}$ **then**
      Update $\mathrm{x}[i]^{k+1}$ using (18)
    else
      Update $\mathrm{x}[i]^{k+1}$ using (19)
    end if
  end for
  for $\{i, j\} \in \mathcal{E}$ **do**
    $\theta = \max\left(0.5, 1 - \frac{\lambda W_{ij}}{\rho |\mathrm{X}[i]^{k+1} + u_{ij}^k - \mathrm{X}[j]^{k+1} - u_{ji}^k|}\right)$
    $z_{ij}^{k+1} = \theta(\mathrm{x}[i]^{k+1} + u_{ij}^k) + (1 - \theta)(\mathrm{x}[j]^{k+1} + u_{ji}^k)$
    $z_{ji}^{k+1} = \theta(\mathrm{x}[j]^{k+1} + u_{ji}^k) + (1 - \theta)(\mathrm{x}[i]^{k+1} + u_{ij}^k)$
    $u_{ij}^{k+1} = u_{ij}^k + (\mathrm{x}[i]^{k+1} - z_{ij}^{k+1})$
    $u_{ji}^{k+1} = u_{ji}^k + (\mathrm{x}[j]^{k+1} - zji^{k+1})$
  end for
until stopping criterion is satisfied

Output: labels $\hat{x}^{(k)}[i]$ for all $i \in \mathcal{V}$

4 Experimental Setup

In this section, we present the different datasets on which we conduct the benchmark as well as the methodology to evaluate the performance of the learning algorithms. More specifically, we discuss in depth how the graph structures and signals are obtained from the data, the parameter settings for the algorithms, the performance metrics used and the GraphX platform configuration.

4.1 Datasets

For the benchmarking process, we select three of the most common types of graphs, namely chains (e.g. time series graphs), grids (e.g. image graphs) and scale-free networks (e.g. social network graphs). For each of these types, we present both synthetic and real data graphs and signals. The networks exhibit a wide variety of characteristics: their size ranges from several hundreds of thousands to millions of nodes and edges, some present edge weights while others (i.e. electricity consumption and Amazon graphs) are unweighted, average node degrees range from 1.8 up to 20 and while some are clearly clustered, others are not. Also, for the real electricity and Amazon graphs some signal values are missing. The main features of these datasets are summarized in Table 1, and in what follows we describe each of this datasets in detail.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	Signals	Weights	Missing $x[\cdot]$	Avg. degree	Clust. coeff.
Synth chain	$1 \cdot 10^6$	$1 \cdot 10^6$	$\{1, 5\}$	$\{1, 2\}$	No	1.99	0.0
Electrcity	$2 \cdot 10^6$	$2 \cdot 10^6$	$\mathbb{R}^+$	$\{1\}$	$1,25\%$	1.99	0.0
Synth grid	$1 \cdot 10^6$	$1.9 \cdot 10^6$	$\{1, 5\}$	$\{1, 2\}$	No	3.99	0.0
Image	$2.7 \cdot 10^5$	$5.38 \cdot 10^5$	$\{-1, 1\}$	$\mathbb{R}^+$	No	3.99	0.0
Synth LFR-L	$1 \cdot 10^5$	$9.45 \cdot 10^5$	$\mathbb{R}^+$	$\mathbb{R}^+$	No	18.99	0.62
Synth LFR-H	$1 \cdot 10^5$	$9.49 \cdot 10^5$	$\mathbb{R}^+$	$\mathbb{R}^+$	No	18.91	0.75
Amazon	$5.24 \cdot 10^5$	$1.7 \cdot 10^6$	$\frac{1}{2}\{0, 2, 3.., 10\}$	$\{1\}$	40%	6.57	0.20
3D road map	$3.97 \cdot 10^5$	$3.77 \cdot 10^5$	$\mathbb{R}$	$\mathbb{R}^+$	No	1.89	$3.69 \cdot 10^{-6}$

Table 1: Main features of the data graphs used to benchmark the GSR algorithms.

Chain Graphs

Chain graphs are characterized by a particular distribution of node degrees and arrangement of edges. In this type of regular graphs, the nodes are placed sequentially and edges connect successive nodes. This particular distribution guarantees that the first and last nodes of the chain will have degree 1 while the rest of nodes will have degree 2. The average degree of chain graphs containing many nodes is ≈ 2, while the clustering coefficient is 0, since the neighbours of every node are never connected to each other.

Synthetic chain graph: The first dataset considered corresponds to a synthetic chain graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, containing a total of $N = 10^6$ nodes $\mathcal{V} = \{1, 2, \ldots, N\}$, which are connected by $N-1$ undirected edges $\mathcal{E} = \{\{i, i+1\}\}_{i=1,\ldots,N-1}$. We assume a partition $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \}$ of the chain graph $\mathcal{G}$, constituted by clusters $\mathcal{C}_l$ of 10 consecutive nodes, i.e.,

$$\mathcal{C}_1 = \{1, \ldots, 10\}, \mathcal{C}_2 = \{11, \ldots, 20\}, \ldots, \mathcal{C}_{N/10} = \{N-9, \ldots, N\}. \tag{20}$$

The weights $W_{i,j}$ for the edges $\{i, j\} \in \mathcal{E}$ are chosen as:

$$W_{i,j} = \begin{cases} 2 & \text{if } i, j \in \mathcal{C}_l \text{ for some } l \in \{1, \ldots, N/10\} \\ 1 & \text{else} \end{cases} \tag{21}$$

Thus, edges within a cluster have weight 2 and those connecting two different clusters have weight 1. We have generated the signals $x[i]$ for the nodes in each cluster $\mathcal{C}_l$ to be constant and equal to either 1 or 5. If the cluster in the chain (20) is in an odd position, the node signal values will be 1, otherwise, the node signals will be 5. This leaves an obvious distribution of node degrees of 50% ones and 50% fives.

In Figure 4 we provide a representation of the chain graph $\mathcal{G}$ and node signals. In this case, only 5 nodes are shown for each cluster $\mathcal{C}_l$ and only two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$.

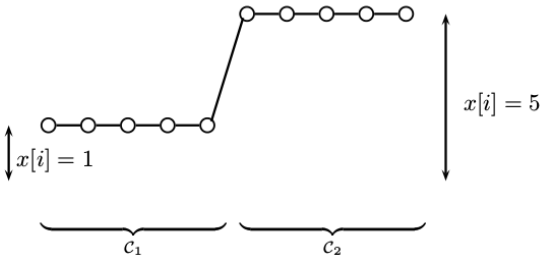

Figure 4: A chain graph partitioned into 2 clusters. The vertical position of the nodes indicates their signal values $x[i]$.

Power consumption graph: The second dataset also conforms to a chain graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, which is, however, generated using real world data freely available from the UCI machine learning repository [28]. This dataset contains measurements of electric power consumption in a private household. The data has been collected over a period of four years at one-minute intervals and contains a total of $N = 2075259$ instances. Nodes $i \in \mathcal{V}$ represent measurements at a specific time instants t_i. The undirected edges $\mathcal{E} = \{\{t_i, t_{i+1}\}\}_{i=1,\ldots,N-1}$ of the graph connect consecutive time instants and have constant weights $W_{i,j} = 1$. The total number of edges is $|\mathcal{E}| = 2075258$.

Each node $i \in \mathcal{V}$ has an associated signal value $x[i] \in \mathbb{R}$ representing the average electrical consumption (in kilowatt) at time t_i. Some nodes in the dataset, around 1,25%, present missing measurements so their signal values are considered to be -1. The distribution of the signal values is presented in Figure 5, where it can be observed that most signal values are in the range 0-4.

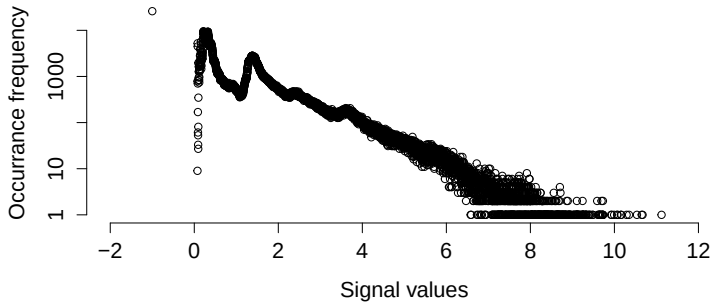

Figure 5: Distribution of signal values on the electricity consumption dataset.

Grid Graphs

The particular characteristics of grid graphs are their arrangement of nodes and degree distribution. The nodes are placed creating a regular mesh, which can be rectangular or square. Each node $i \in \mathcal{V}$ is connected to its up to 4 neighbours (upper, lower, left and right), so the nodes in the interior of the grid have a fixed degree on 4. The nodes in the corners present degree 2, while the remaining nodes have degree 3. This results in an average degree of ≈ 4 for grid graphs containing a large number of nodes. Due to the 4-connected neighbourhood, the clustering coefficient is 0.

Synthetic grid graph: We generate a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ containing $N = 10^6$ nodes. These nodes $\mathcal{V} = \{1, 2, \ldots, N\}$ are arranged in a 1000×1000 elements grid where each node is connected to up to four neighbours. This gives a total of $|\mathcal{E}| = 1998000$ edges. The graph $\mathcal{G}$ is partitioned into a set of 100 disjoint clusters so that $\mathcal{F} = \{\mathcal{C}_1, \ldots.., \mathcal{C}_{100}\}$. Each cluster $\mathcal{C}_l$ is a sub-grid of 100×100 nodes.

The weights $W_{i,j}$ for the edges $\{i, j\} \in \mathcal{E}$ are chosen using the same strategy presented for the synthetic chain graph, i.e.

$$W_{i,j} = \begin{cases} 2 & \text{if } i, j \in \mathcal{C}_l \text{ for some } l \in \{1, \ldots, 100\} \\ 1 & \text{else} \end{cases} \tag{22}$$

The signals $x[i]$ associated with each node $i \in \mathcal{V}$ are set to be either 1 or 5. The signals of nodes belonging to the same cluster $\mathcal{C}_l$ are identical and adjacent clusters always have different node signals. This results in a checkboard pattern similar to the representation in Figure 6 with a 50-50 distribution of signal values.

Image graph: Our next grid graph is generated from a 600×450 pixels image obtained from the "grabCut" dataset [35]. Each pixel i of the image is represented by an RGB vector $v[i] = (\text{red}[i], \text{green}[i], \text{blue}[i]) \in \{0, 1, ..., 255\}^3$. A grid graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ is constructed such that nodes represent individual pixels and pixels are connected to their four closest neighbours. There are a total of $N = 270000$

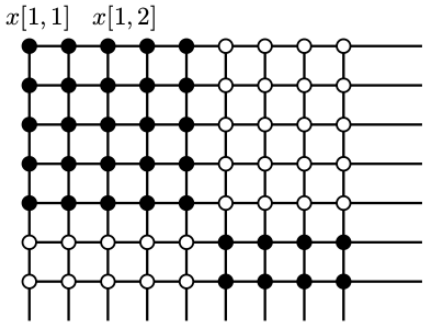

Figure 6: Representation of the synthetic grid graph and node signals. Nodes are shaded according to their signal value, black if $x[i] = 1$ and white if $x[i] = 5$.

nodes connected by $|\mathcal{E}| = 538950$ edges. The edge weights are computed using the Euclidean distance between the RGB components of connected pixels as follows:

$$W_{i,j} = exp(-(1/\sigma)||v[i] - v[j]||) \tag{23}$$

Where σ is defined as:

$$\sigma = median\{||v[i] - v[j]||\}_{\{i,j\}\in\mathcal{E}} \tag{24}$$

The signal values $x[\cdot]$ label each node $i \in \mathcal{V}$ as belonging to the foreground of the image ($\mathcal{R}1$) or to the background ($\mathcal{R}2$). These regions are *a priori* known due to manual labelling. The node signals are set so that: $x[i] = 1\ \forall i \in \mathcal{R}1$ and $x[i] = -1\ \forall i \in \mathcal{R}2$. This, in turn, gives a distribution of signal values of 80% and 20% respectively.

In Figure 7 we show the image represented as a grid graph as well as the boundary between the foreground ($\mathcal{R}1$) and background ($\mathcal{R}2$).

Figure 7: Real image used as grid graph. The white line delimits the boundary between foreground (flower) and background (rest of the image).

Power Law Graphs

The power law graphs used in our experiments exhibit a diversity of average degrees and clustering coefficients. In particular, the synthetic LFR graphs present very densely connected nodes and high clustering coefficients. The 3D road dataset presents low average degree and clustering coefficient, while the Amazon dataset contains intermediate values in both cases.

Synthetic LFR graph In order to generate synthetic scale-free networks, we use the popular LFR model previously introduced in Section 2. The LFR model allows us to create synthetic community structured graphs whose characteristics perfectly mimic those of real world networks. Some of these characteristics are power law distribution of node degrees and community sizes and different inter/intra cluster edge weights and densities. There are several parameters that can be specified when generating an LFR graphs such as: number of nodes (N), average degree (k), maximum degree ($maxk$), minimum ($minc$) and maximum ($maxc$) community sizes, mixing parameter for the weights (muw), average clustering coefficient (C) etc. Also, communities can be generated with or without overlapping, i.e. nodes can belong to a single community or several at the same time. An example of a network that could be obtained using the LFR modes is shown in Figure 8.

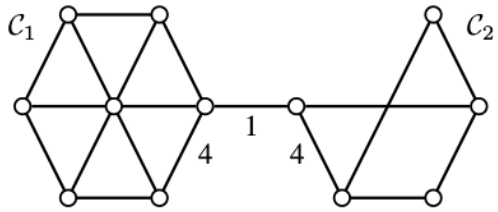

Figure 8: Subgraph of graph $\mathcal{G}$ and node signal values.

We generate two graphs with disjoint communities (no overlapping nodes) with relatively low (LFR-L) and high (LFR-H) average clustering coefficients. These coefficients correspond to the maximum and minimum values we could obtain using the LFR model and are respectively, $C = 0.62$ and $C = 0.75$. These values are also close to the average clustering coefficient expected for complex networks of $C \approx 3/4$ ([41]). In addition to the clustering coefficients, we specify the minimum number of parameters in order to generate the networks and allow the model to infer the rest. These parameters are: $N = 10^5$, $k = 20$, $maxk = 300$, $minc = 1000$ and $muw = 0.1$.

The two graphs obtained have approximately the same number of nodes $N = 10^5$ and edges $|\mathcal{E}| \approx 945000$. The edge weights $W_{i,j}$ are positive numbers in $\mathbb{R}$ and, on average, higher for inter-cluster edges than intra-cluster edges. Both graphs are partitioned in a set of 1399 disjoint clusters or communities, i.e. $\mathcal{F} = \{\mathcal{C}_1, \ldots.., \mathcal{C}_{1399}\}$. Each cluster $\mathcal{C}_l$ contains a set of nodes whose signal value $x[i]$ is $x[i] = l \ \forall i \in \mathcal{C}_l$. The distributions of the node degrees d_i for both graphs are presented in Figures 9 and 11. As expected, both graphs follow a log-normal/power law composite. In Figures

10 and 12, the occurrences of the different signal values $x[i]$ are depicted for each LFR graph and the power law distribution of community sizes can also be observed.

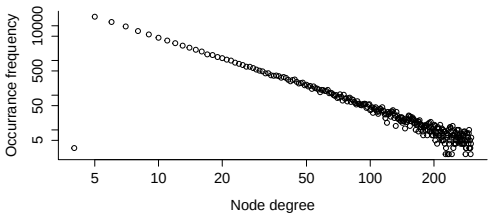

Figure 9: Degree distribution of the LFR-L graph.

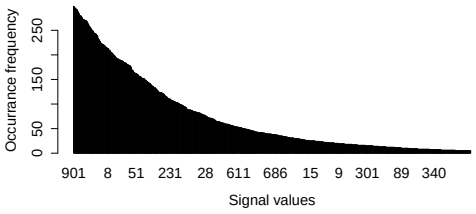

Figure 10: Distribution of LFR-L graphs signal values.

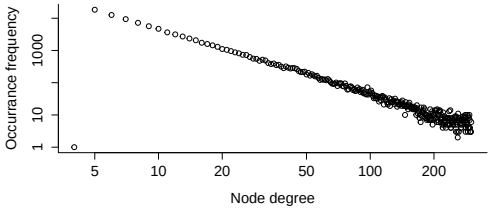

Figure 11: Degree distribution of the LFR-H graph.

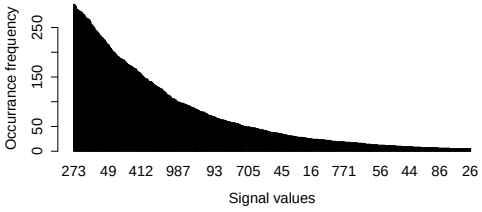

Figure 12: Distribution of LFR-H graphs signal values.

Amazon graph: Our first real, scale-free network is generated from a dataset of products once available at the online retail shop Amazon.com. The dataset can be obtained from the Stanford network analysis platform (SNAP)[27]. Starting from the raw data, we first construct a preliminary graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathbf{W}')$ where each node $i \in \{1, ..., N\}$ represents a product and two nodes i and j are connected by an edge $e = \{i, j\} \in \mathcal{E}'$ if i is co-purchased with product j or vice versa. The graph $\mathcal{G}'$ is unweighted, thus $W'_{i,j} = 1$ for all $\{i, j\} \in \mathcal{E}'$. The average rating of product i is regarded as the graph signal $x[i]$, with values from the set $\mathcal{A} = \frac{1}{2}\{2, 3, ..., 10\}$. Approximately 40% of the nodes $i \in \mathcal{V}'$ have unknown signal values so they are initialized to 0.

The obtained graph $\mathcal{G}'$ is disconnected so we restrict our analysis to its largest connected component which spans over 90% of the nodes. Said largest component yields a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ with $|\mathcal{V}| = 524366$ nodes, out of which 311641 have known labels, and which are connected by $|\mathcal{E}| = 1723624$ edges. The power law distribution of the node degrees d_i can be observed in Fig. 13. Most of the known signal values are in the range 4-5 as presented in Fig. 14.

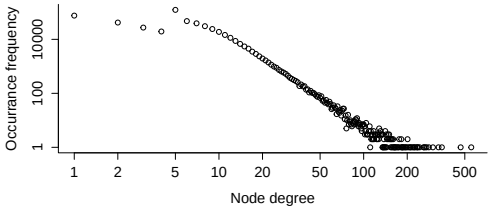

Figure 13: Distribution of node degrees in log-log scale for the Amazon graph $\mathcal{G}$.

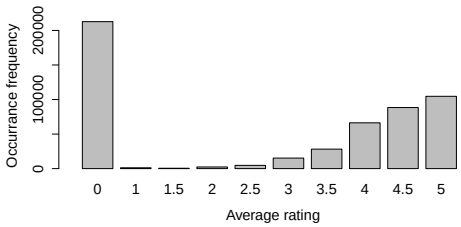

Figure 14: Distribution of node signal values for the Amazon graph $\mathcal{G}$.

3D road network graph: The last power law graph is generated from a dataset available at the UCI Machine Learning Repository [28] describing a 3D road network from Denmark. The dataset was generated by adding accurate elevation information over a 2D road network. A more detailed description of this process can be found in [29]. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ has been obtained from the raw data where nodes $i \in \mathcal{V}$ represent the location of intersections or ends of roads and edges $\{i, j\} \in \mathcal{E}'$ represent segments of road. The graph $\mathcal{G}$ contains several disconnected components and a total of $N = 397978$ nodes and $|\mathcal{E}| = 377545$ edges. The edge weight $W'_{i,j}$ for each $\{i, j\} \in \mathcal{E}$ is calculated using the geodesic distance between the latitude and longitude of nodes i and j. In particular, these weights measured in kilometres, are computed using the Haversine formula [9].

The signal values $x[i]$, associated to each node $i \in \mathcal{V}$, correspond to the altitude measured in meters at the location represented by i. The signals $x[i] \in \mathbb{R}$ can be negative, indicating that the location is below sea level. The dataset has no missing altitude information for any node. The distribution of node degrees is, as expected due to the nature of the data graph, limited to a very low value, as shown in Figure 15. The distribution of the signal values (Fig. 16) indicates that most nodes have an altitude slightly below sea level.

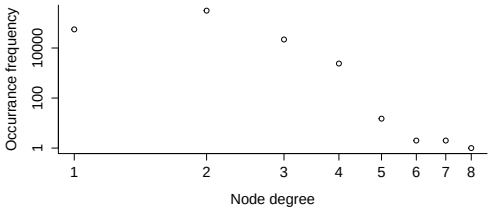

Figure 15: Distribution of node degrees in log-log scale for the 3D road network.

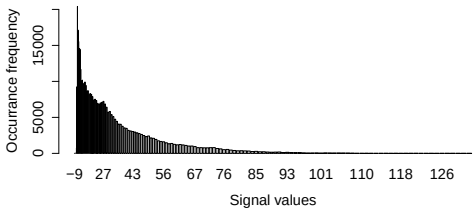

Figure 16: Distribution of node signal values for the 3D road network.

4.2 Evaluation Methodology

We start this subsection by discussing the parameter configuration for the algorithms presented in Section 3. We used a fixed number of 100 iterations. This value is expected to be sufficiently large, provided the theoretical convergence rate of the algorithms of approximately $1/k$. The second global parameter is the sampling set $\mathcal{M}$, of initially known signal values $x[i] = x_i$. Its size varies from 10% up to 80% of the graph nodes with known signals and is selected according to different strategies. This set $\mathcal{M}$, is the same for each algorithm on each independent experiment. The rest of the signals $x[i]$ for $i \in \mathcal{V} \backslash \mathcal{M}$ are set to 0 and expected to be recovered by the algorithms. Methods such as SLP and nLasso present additional parameters that can be tuned. In this case, we use the configurations recommended by the authors in the original papers with no additional tuning. Specifically, for SLP we use $\lambda = 1$ and for nLasso $\lambda = 1$ and $\rho = 0.1$.

Performance Metrics

In order to evaluate the algorithms on each dataset presented in Table 1, we have designed two classes of experiments. For the first class, we compare the behaviour of the algorithms on the real against the synthetic datasets. For the second class, we perform several experiments exclusively on the synthetic data in order to test: the sensitivity to noise of the methods, the effect of the weights, sampling set selection strategy and sampling set size as well as the effect on the scalability of the method of increasing cluster sizes. Below, we describe this experiments more in detail organized by the performance metrics they measure.

Accuracy: We measure the accuracy of the algorithms in terms of normalized MSE between the known ($x[\cdot]$) and the predicted ($\hat{x}[\cdot]$) signal values for each dataset. The NMSE is computed as $\varepsilon = \|\hat{x}^{(k)}[\cdot] - x[\cdot]\|_2^2 / \|x[\cdot]\|_2^2$ and presented w.r.t. the iteration number k. We also compare the accuracy achieved by the algorithms on the synthetic graphs to the one obtained on the real ones.

Sampling Sets: We study the NMSE of the algorithms for varying sizes of the sampling set $\mathcal{M}$, ranging from 10% to up to 80% of the known signal values for each graph. Moreover, we compare a random selection of initial samples against a cluster based selection. We perform this comparison only on the synthetic graphs which present a well-defined clustering structure. In both cases, 10% of graph nodes are selected as part of the sampling set $\mathcal{M}$, in the first case using a random selection and in the second taking an approximately equal amount of nodes from each of the graph clusters $\mathcal{C}_l$.

Scalability: For each algorithm, we study its scalability in terms of the execution time. We use the synthetic graphs to compare the algorithms on different cluster configurations, specifically with 1,2,4 and 8 worker nodes.

Weighted/Unweighted graphs: We further analyse the differences in NMSE for the same synthetic graphs generated with and without weights. The aim of this experiment is to determine to what extent the algorithms rely on the information contained in the edge weights.

Noise robustness: The algorithms behaviour in case of noise is analysed also on the synthetic graphs. We add normally distributed Gaussian noise ($N(0,1)$) to each data point x_i for all $i \in \mathcal{V}$. The noisy signal values are thus generated from a distribution $s_i \sim \mathcal{N}(x_i, 1)$.

Cluster Configuration

We benchmark all the GSR methods on an identical hardware configuration. Specifically, we use a single server machine with a 64 bit Intel® Xeon® CPU E3-1230 clocked at 3.20GHz (providing 4 physical cores and 8 threads) 16 GB of RAM and 10 TB of disk. The software environment consists of the Ubuntu 16.04.2 LTS operating system, Java JDK 1.8 (8u131), Eclipse IDE, Scala version 2.10, Apache Maven 3.0.5 and Spark and GraphX version 2.0.1. Other dependencies required are Apache Hive, Mllib and SparkSQL all in version 1.6.1.

As mentioned in Section 2, three cluster managers are available in Spark. For our experiments, we have selected the standalone cluster manager. This mode allows us to deploy a whole cluster on a single machine where each CPU thread becomes a cluster node. Therefore, we can have up to 8 worker nodes configurations one of them being at the same time worker and master node. The workers communicate through the network stack simulating a distributed environment, thus the obtained results are comparable to those of a real distributed setting (accounting for some transmission time between machines). Also, by using a single machine deployment we limit potential network related factors that could affect the algorithms scalability results such as cable category and length, traffic on the network or inconsistent machine behaviour. In addition, the focus here is on the algorithm rather than the platform scalability. Furthermore, graphs with tens of millions of nodes and edges can be analysed on this cluster configuration which is sufficient for the intended benchmarking. The ideal number of Spark partitions assigned to each worker node has been experimentally found to be 2.

5 Results and Discussion

Along this section, we present and discuss the experimental results obtained. Said results are organized by the type of graphs i.e. chains, grids and power-law graphs.

5.1 Chain Graphs

First, we compare the recovery on the synthetic and real chain graphs using a randomly selected sampling set $\mathcal{M}$ containing, in each case, 20% of the graph nodes. For a visual comparison, we depict in Figures 17 and 18 the true $x[\cdot]$ and recovered signal values $\hat{x}[\cdot]$ for the first 100 nodes of each graph. In both cases, the recoveries of SLP (depicted in green) and LP_sr (depicted in red) are similar to each other and the closest among all methods to the real signal value (shown in black).

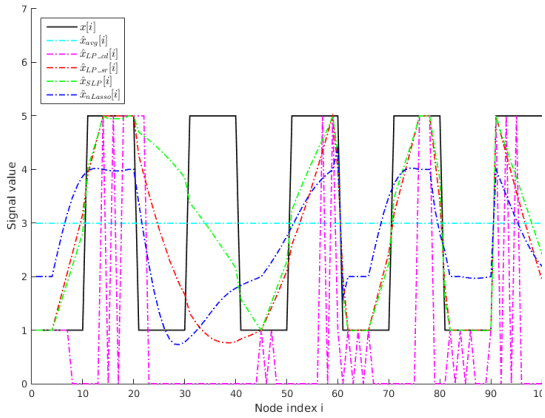

Figure 17: Recovered graph signals for the firs 100 vertices of the synthetic chain graph.

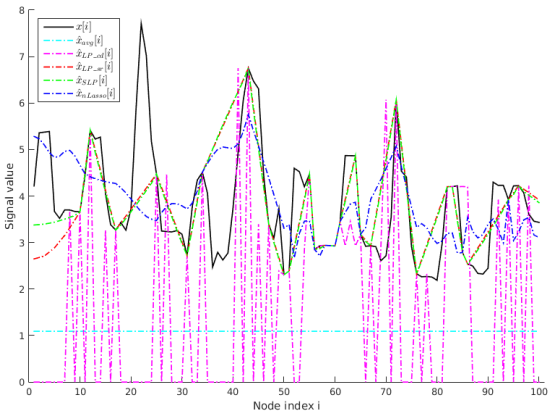

Figure 18: Recovered graph signals for the firs 100 vertices of the electricity consumption graph.

We also present the accuracy in terms of NMSE w.r.t the iteration number for each graph in Figures 19 and 20. The methods based on a non-iterative recovery process, i.e. Avg_cons and LP_cd, display a constant curve representing their final NMSE value. The rest of the methods converge in a similar fashion, although SLP achieves the lowest error rate. The final error rates after 100 iterations for each method can be better observed in Figure 21.

For this specific type of synthetic chain graph and with only 10% of the graph nodes in the sampling set $\mathcal{M}$, a cluster-based initialization provides much better results than a random one (Fig. 22). When increasing the size of the sampling set $\mathcal{M}$ from 10% up to 80% of the nodes, the NMSE, as expected, decreases. Proportionally, the method that benefits the most from this increase is SLP as depicted in Figure 23.

The execution times for different cluster configurations (1-8 worker nodes) are shown in Figure 24. It can be seen that while LP_cd is the fastest method, SLP is the slowest. Additionally, when the number of worker nodes of the cluster is increased, all methods exhibit the same proportional decrease in execution time, and

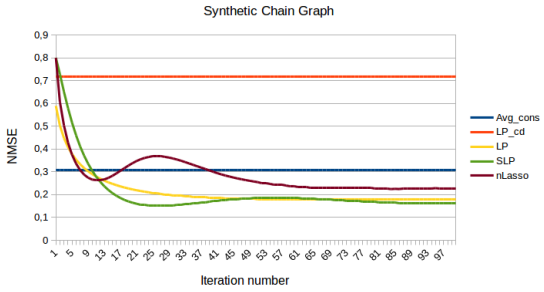

Figure 19: NMSE against iteration number for the synthetic chain graph.

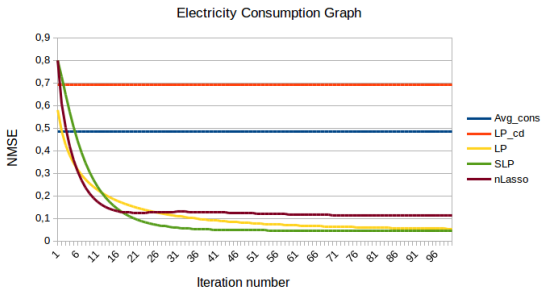

Figure 20: NMSE against iteration number for the electricity consumption graph.

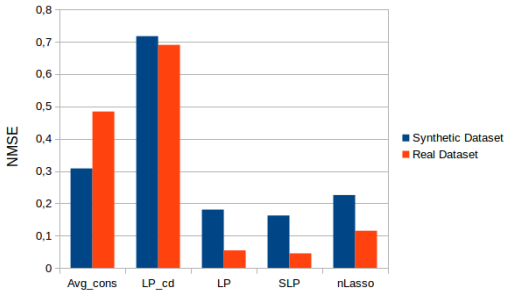

Figure 21: Comparison of the NMSE after 100 iterations achieved by each algorithm, for the real and synthetic graphs.

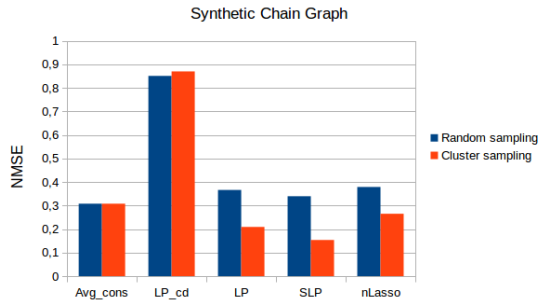

Figure 22: Error rates using random and cluster-based initialization.

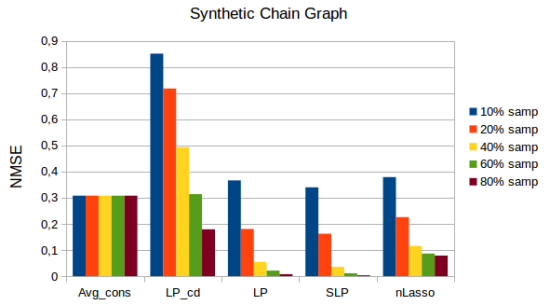

Figure 23: Evolution of the error rates for different sampling set sizes.

in particular, the same as LP_cd. The main computation in this method is the LPA algorithm, whose scalability has been extensively studied by the developers at the GraphX project. For this reason, we can conclude that all the methods presented are highly scalable on chain graphs.

The lack of weights supporting the topology of the synthetic chain graph affects, in particular, the accuracy of SLP (Fig. 25). This method relies heavily on the weights as a tool to determine the graph partitioning $\mathcal{F}$. Other methods such as Avg_cons and LP_cd do not use the smoothness of the graph signal so they are not

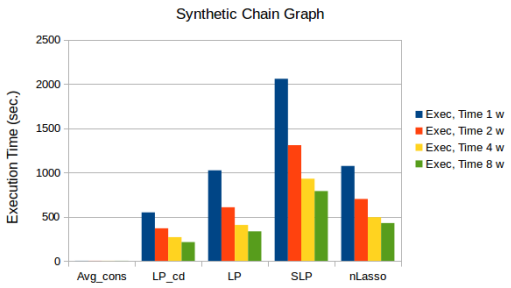

Figure 24: Execution times for different cluster configurations.

affected by the lack of weights. The sensitivity to noisy signal values for all methods is similar as indicated by Fig. 26.

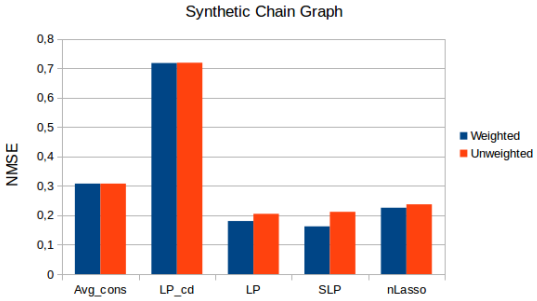

Figure 25: Error rates for weighted and unweighted graphs.

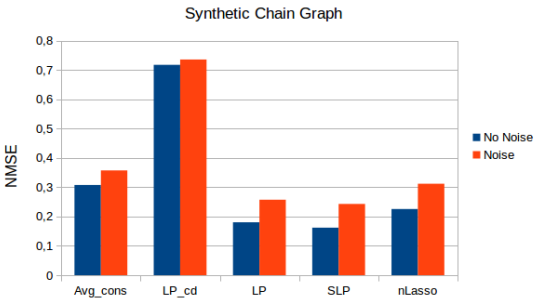

Figure 26: Error rates for graph signals with and without random Gaussian noise added.

5.2 Grid Graphs

For the synthetic grid graph we depict as black pixels the nodes whose signal value is 1 and as white pixels those whose signal value is 5. We use a grey scale for intermediate values. The true labels and best recovery are presented in Fig. 27, 28. We compare these results to the recovery of the flower image dataset where nodes with signal values -1 are represented in black and those with signal values 1 are shown in white, using again a grey scale for intermediate values (Fig. 29 and 30). In both cases, we use a randomly selected sampling set of 20% of the graph nodes. For the synthetic grid graph, SLP is the method that performs the best closely followed by LP. However, SLP returns sharper borders between the black and white squares than LP does. For the real image, on the other side, is LP the method that obtains the best recovery. For the remaining figures, not presented here, we refer the reader to Appendix A (Fig. A1-A8).

The convergence rates of the algorithms for the above-mentioned experiments are shown in Figures 31 and 32. The convergence of the three iterative algorithms

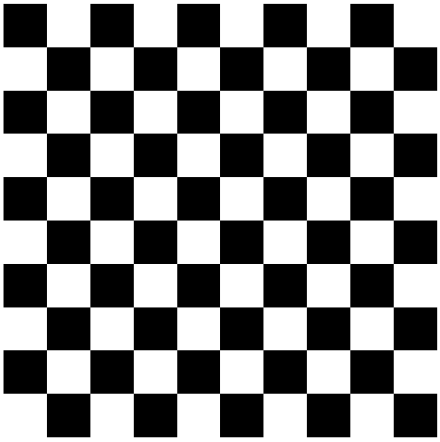

Figure 27: True graph signal with a checkboard pattern.

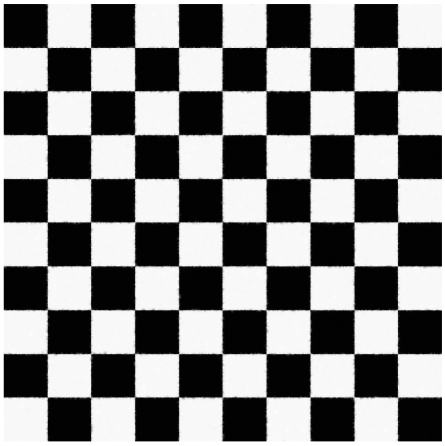

Figure 28: Recovered graph signal using SLP algorithm.

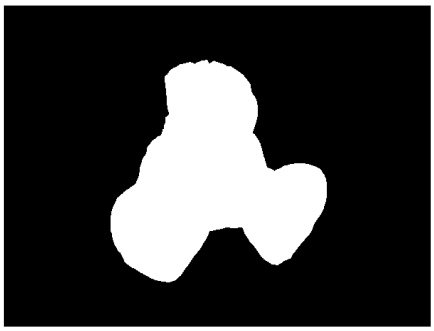

Figure 29: True signal values for the flower image dataset.

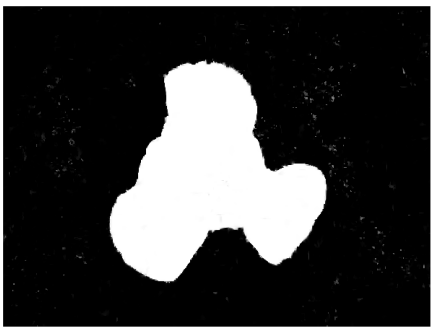

Figure 30: Recovered graph signal using LP_sr algorithm.

(LP_sr, SLP and nLasso) is, in this case, very similar. The NMSEs of the methods on both real and synthetic data are very similar, as indicated in Fig. 33.

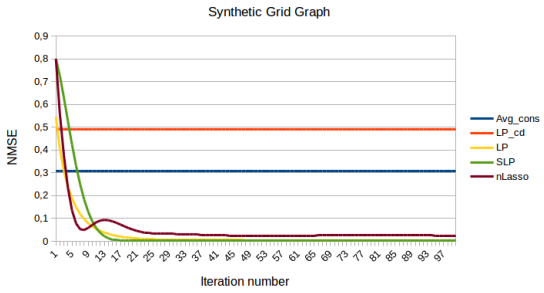

Figure 31: NMSE against iteration number for the synthetic grid graph.

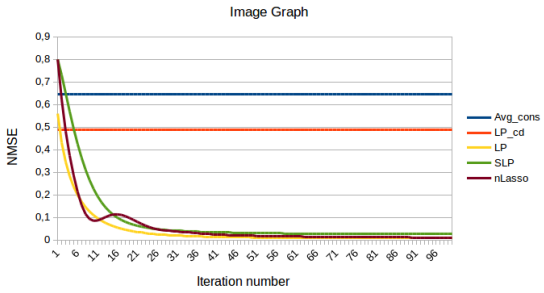

Figure 32: NMSE against iteration number for the real image graph.

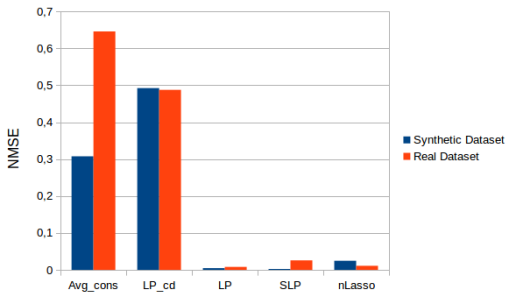

Figure 33: Comparison of the NMSE achieved by each algorithm after 100 iterations, for the real and synthetic graphs.

In Figure 34 we compare a randomly selected initial set $\mathcal{M}$ of 10% of the graph nodes to the cluster-based initialization presented in Subsection 4.2. In this case, due to the large size of the clusters and identical shape, no significant differences in the final NMSEs of the algorithms can be found between the two sampling set selection strategies. For sampling sets $\mathcal{M}$ of increasing sizes, the error rates decrease, following the same patterns as for chain graphs (Fig. 35).

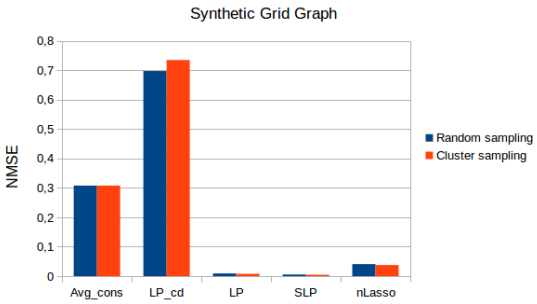

Figure 34: Error rates using random and cluster-based sampling set initialization.

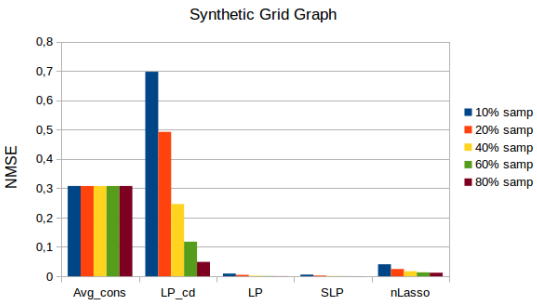

Figure 35: Evolution of the error rates for different sampling set sizes.

The execution times for different cluster configurations (1-8 worker nodes) are depicted in Figure 36. These execution times decrease proportionally for all methods and in a similar pattern as for chain graphs. We can, therefore, conclude that also for grid graphs the methods implemented scale well.

The lack of weights on the synthetic grid graph has almost no effect on the accuracy of the algorithms (Fig. 37). This effect can be explained due to the large cluster sizes which, in turn, results in smaller amounts of inter-cluster edges making them less relevant for the recovery problem. Noise, however, affects all methods, and in approximately the same proportions as in the experiments on chain graphs (Fig. 38).

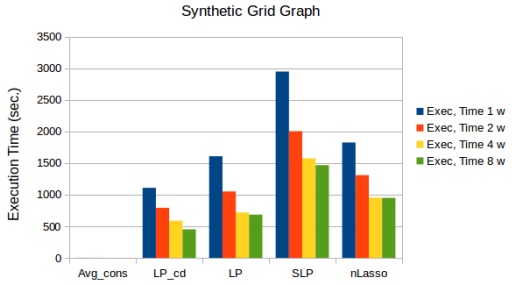

Figure 36: Execution times for different cluster configurations.

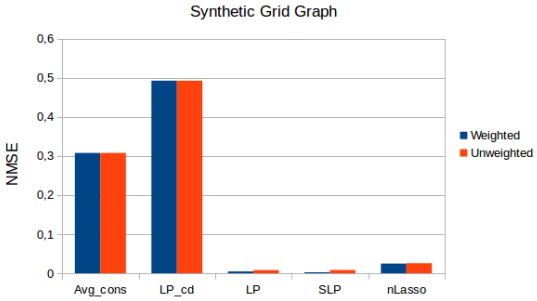

Figure 37: Error rates for weighted and unweighted grid graphs.

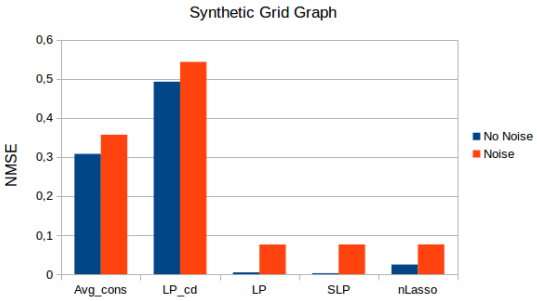

Figure 38: Error rates for graph signals with and without random Gaussian noise added.

5.3 Power Law Graphs

The convergence rates of the algorithms on the synthetic LFR datasets with low and high clustering coefficients are very similar (Fig. 39 and 40). Although nLasso is the method that converges the slowest, is also the one that achieves in both cases the lowest NMSE after 100 iterations. We also highlight that, due to the clear community structure of the datasets, LP_cd performs very well. On the other hand, for the real Amazon and 3D road networks, is SLP the one that converges to the lowest error rate (Fig. 41 and 42). In this case, with less clear clusters, the community-based labelling does not provide a good signal recovery. In Figure 43 we compare the NMSE after 100 iterations of all algorithms on each real and synthetic graph. The sampling sets used contained 20% of the nodes with known signal values in each case.

No significant differences are found for the synthetic power law graphs between using a random or a cluster-based sampling set selection strategy (Fig. 44 and 45). Under initializations with different sampling set sizes nLasso is the method that increases its accuracy the most (Fig. 46 and 47).

For both LFR synthetic graphs, the execution times decrease as the number of workers is increased following roughly the same proportions seen in chain and grid graphs (Fig. 48 and 49). We can thus conclude that the scalability of the methods is not influenced by the topology of the underlying graph.

Removing the weights from these synthetic graphs has a very negative effect on the

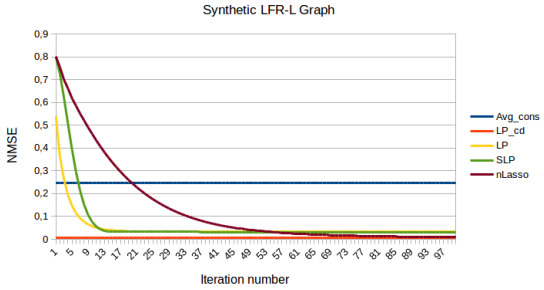

Figure 39: NMSE against iteration number for the synthetic LFR-L graph.

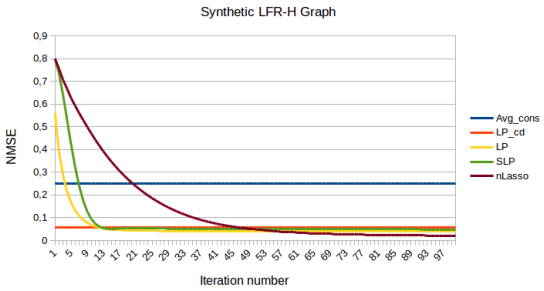

Figure 40: NMSE against iteration number for the synthetic LFR-H graph.

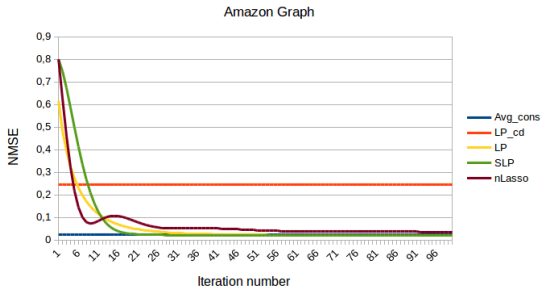

Figure 41: NMSE against iteration number for the Amazon graph.

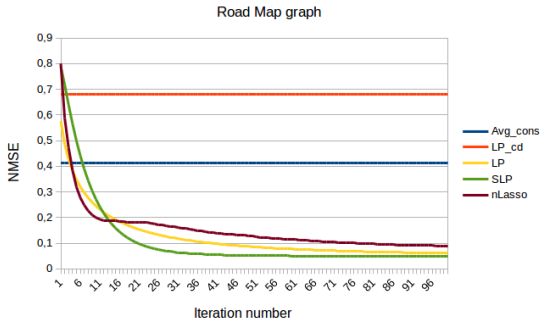

Figure 42: NMSE against iteration number for the 3D road graph.

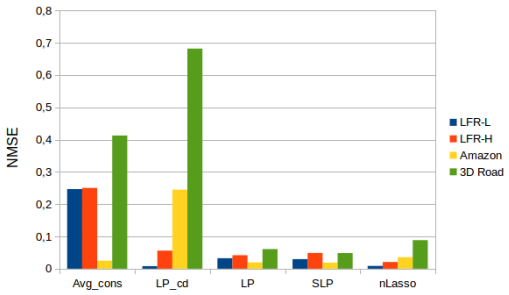

Figure 43: Comparison of the NMSE achieved by each algorithm after 100 iterations, for the real and synthetic power law graphs.

accuracy of nLasso. The rest of the methods seem more robust to this perturbation as seen in Figures 50 and 51. Moreover, all algorithms have proven high robustness against random Gaussian noise in this scenario (Fig. 53 and 53).

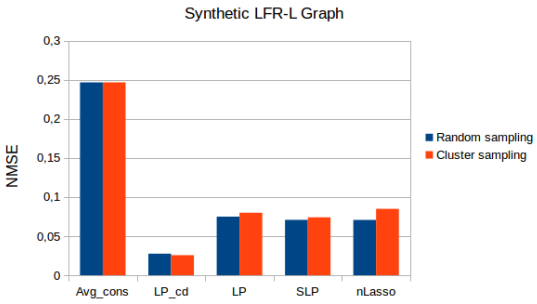

Figure 44: Error rates using random and cluster-based initializations for the LFR-L graph.

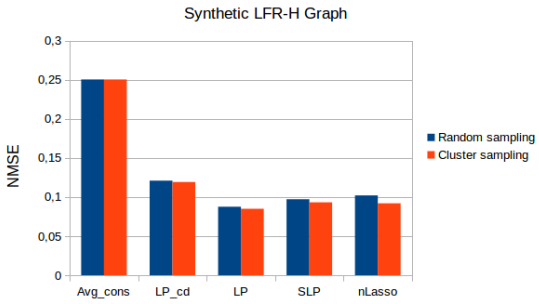

Figure 45: Error rates using random and cluster-based initializations for the LFR-H graph.

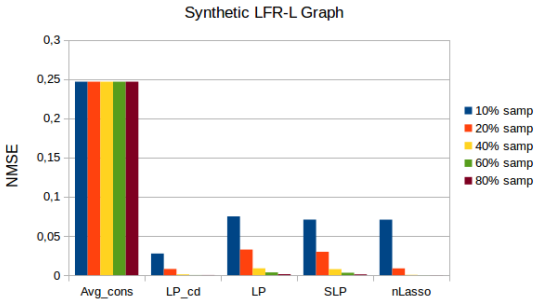

Figure 46: Evolution of the error rates for different sampling set sizes on the LFR-L graph.

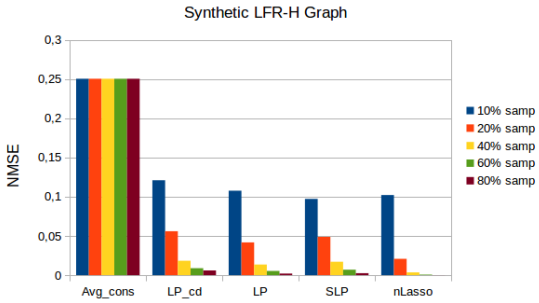

Figure 47: Evolution of the error rates for different sampling set sizes on the LFR-H graph.

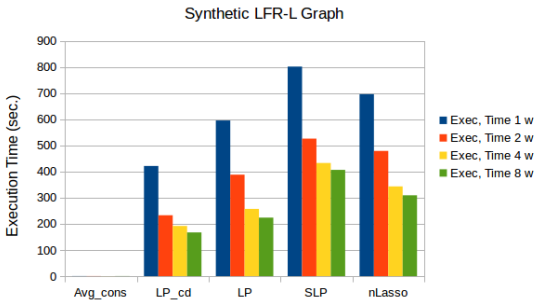

Figure 48: Execution times for different cluster configurations on the LFR-L graph.

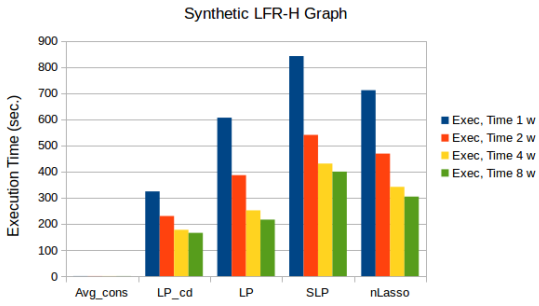

Figure 49: Execution times for different cluster configurations on the LFR-H graph.

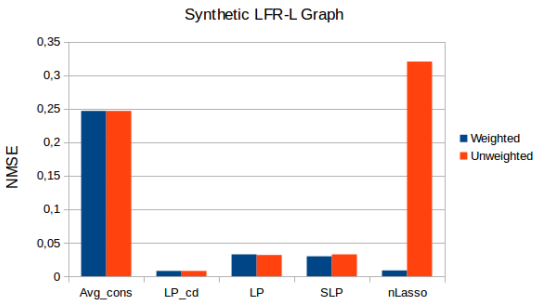

Figure 50: Error rates for weighted and unweighted graphs on the LFR-L graph.

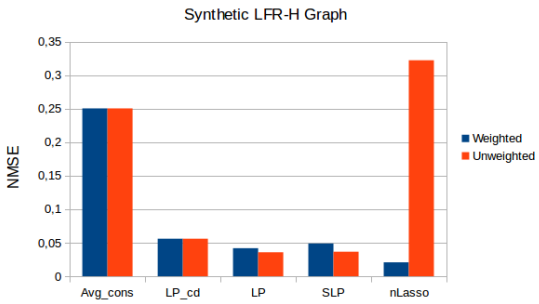

Figure 51: Error rates for weighted and unweighted graphs on the LFR-H graph.

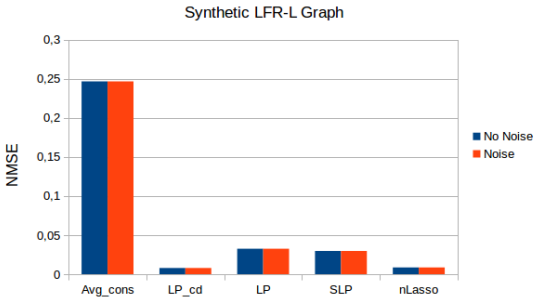

Figure 52: Error rates on the LFR-L graph for signals with and without random Gaussian noise added.

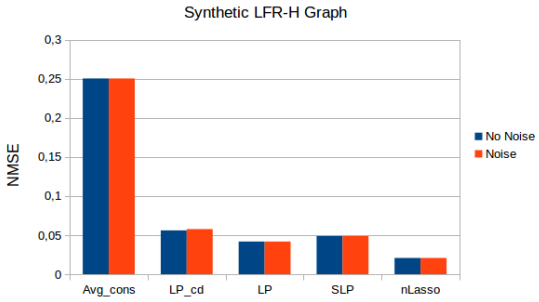

Figure 53: Error rates on the LFR-H graph for signals with and without random Gaussian noise added.

6 Conclusions and Future Work

In this thesis, we have reviewed and empirically compared five algorithms for semi-supervised learning from big data graphs. These methods have been benchmarked on a variety of real graphs constructed from different domains as well as on synthetic ones with diverse topological features. Our experiments show that methods using TV minimization as a smoothness measure perform marginally better than those relying on the quadratic form. In particular, SLP presents the lowest generalization error on regular as well as power law graphs emerging from real data. Furthermore, nLasso also based on TV minimization is the most accurate on synthetic power law graphs. The labelling method tested based on community detection has shown to perform well for synthetic power law graphs which follow closely the SSL cluster or manifold assumption. Nevertheless, the same method performs rather poorly on all graphs emerging from real data. Although not studied in this thesis, we point out that the accuracy of SLP and nLasso can be further improved by tuning their parameters.

Also from the experiments presented in Section 5, we can conclude that the algorithms exhibit similar convergence rates and that approximately 30 iterations are sufficient to achieve a low generalization error. These convergence rates are independent of the graph size. However, for graphs of the same size, a lower average degree implies slower convergence. If the execution time is a limiting factor, then the method providing the best trade-off between accuracy and execution time is LP. If by contrary, a low computation time is not required, then SLP should be considered since it performs the best in most cases. We can further conclude, based on the experiments, that due to its flexibility (i.e. capacity of expressing other GSSL algorithms) and relatively low complexity and execution time, the most appropriate framework to implement other GSSL methods is ADMM.

Additionally, experiments show that topological feature of the graphs such as clustering coefficient, average degree or edge weights do not have a significant impact on the accuracy of the methods tested. Nevertheless, the sampling set selection strategy does influence the final NMSEs. Specifically, results show that the importance of the sampling set selection strategy increases as the size of the graph clusters decreases. A simple sampling technique, such as selecting nodes from clusters proportionally to their size, results in great improvements in accuracy as seen in the experiments on chain graphs. Furthermore, according to our tests regarding different sampling set sizes, at least 20% of the graph nodes should be sampled in order to obtain a low generalization error.

Topics for future work include adding other algorithms to the benchmark, analysing to what extent the accuracy of SLP and nLasso can be improved by tuning their parameters and increasing the size of the cluster and datasets used. Moreover, we plan to extend our experiments on sampling set selection strategies by including more refined methods such as the one proposed in [**I**] for nLasso.

References

[1] A. Anis, A. Gadde, and A. Ortega. Towards a sampling theorem for signals on arbitrary graphs. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3864–3868, 2014.

[2] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, Oct. 1999.

[3] M. Belkin, I. Matveeva, and P. Niyogi. *Regularization and Semi-supervised Learning on Large Graphs*, pages 624–638. Springer Berlin Heidelberg, 2004.

[4] G. Birkhoff and S. MacLane. *Introduction to Graph Theory.* Longman, 1 edition, 1953.

[5] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 19–26, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[6] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424(4?5):175 – 308, 2006.

[7] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge Univ. Press, Cambridge, UK, 2004.

[8] A. E. Brouwer and W. H. Haemers. *Spectra of Graphs.* New York, NY, 2012.

[9] G. V. Brummelen. Heavenly mathematics: The forgotten art of spherical trigonometry. *Princeton University Press*, Nov. 2015.

[10] A. Chambolle, V. Caselles, D. Cremers, M. Novaga, and T. Pock. An introduction to total variation for image analysis. In *Theoretical foundations and numerical methods for sparse recovery*, volume 9 of *Radon Ser. Comput. Appl. Math.*, pages 263–340. Walter de Gruyter, Berlin, 2010.

[11] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning.* The MIT Press, Cambridge, Massachusetts, 2006.

[12] S. Chen, A. Sandryhaila, J. M. F. Moura, and J. Kovacevic. Signal Recovery on Graphs: Variation Minimization. *ArXiv e-prints*, Nov. 2014.

[13] S. Chen, R. Varma, A. Sandryhaila, and J. Kovacevic. Discrete signal processing on graphs: Sampling theory. *IEEE Transactions on Signal Processing*, 63(24):6510–6523, Dec 2015.

[14] S. Chen, R. Varma, A. Singh, and J. Kovacevic. Signal Recovery on Graphs: Random versus Experimentally Designed Sampling. In *Proc. Int. Conf. Sampling Th. and Applications (SampTA)*, pages 337–341, May 2015.

[15] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[16] S. Cui, A. Hero, Z.-Q. Luo, and J. Moura, editors. *Big Data over Networks*. Cambridge Univ. Press, 2016.

[17] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[18] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst. Learning graphs from signal observations under smoothness prior. arXiv:1406.7842v1, 2014.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[20] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[21] D. Hallac, J. Leskovec, and S. Boyd. Network lasso: Clustering and optimization in large graphs. In *Proc. SIGKDD*, pages 387–396, 2015.

[22] D. Hallac, C. Wong, S. Diamond, R. Sosic, S. P. Boyd, and J. Leskovec. Snapvx: A network-based convex optimization solver. *CoRR*, abs/1509.06397, 2015.

[23] F. Harary. *Graph Theory*. Addison-Wesley Series in Mathematics. Addison Wesley, 1969.

[24] A. Jung, A. O. Hero, III, A. Mara, and S. Jahromi. Semi-Supervised Learning via Sparse Label Propagation. *ArXiv e-prints*, Dec. 2016.

[25] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *J. ACM*, 49(5):616–639, Sept. 2002.

[26] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78:046110, Oct 2008.

[27] J. Leskovec and A. Krevl. Snap datasets: Stanford large network dataset collection.

[28] M. Lichman. UCI machine learning repository, 2013.

[29] C. S. J. Manohar Kaul, Bin Yang. Building accurate 3d spatial networks to enable next generation intelligent transportation systems. *Proceedings of International Conference on Mobile Data Management (IEEE MDM)*, pages 3–6, June 2013.

[30] S. K. Narang, A. Gadde, and A. Ortega. Signal processing techniques for interpolation in graph structured data. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5445–5449, May 2013.

[31] M. E. J. Newman. *Networks: An Introduction*. Oxford Univ. Press, 2010.

[32] T. Pock and A. Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[33] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks, Sept. 2007.

[34] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases*. O'Reilly Media, Inc., 2013.

[35] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, Aug. 2004.

[36] A. Sandryhaila and J. M. F. Moura. Discrete signal processing on graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656, 2013.

[37] A. Sandryhaila and J. M. F. Moura. Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure. *IEEE Signal Processing Magazine*, 31(5):80–90, Sept 2014.

[38] J. Sharpnack, A. Rinaldo, and A. Singh. Sparsistency of the edge lasso over graphs. *AIStats (JMLR WCP)*, 2012.

[39] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, May 2013.

[40] P. P. Talukdar and F. Pereira. Experiments in graph-based semi-supervised learning methods for class-instance acquisition. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1473–1481, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[41] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[42] T. White. *Hadoop: The Definitive Guide*. O'Reilly, 2009.

[43] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*. ACM.

[44] Z. XJ. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison, 2006.

[45] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. 10:10–10.

[46] D. Zhou and B. Schölkopf. A regularization framework for learning from graph data. In *ICML workshop on statistical relational learning and Its connections to other fields*, volume 15, pages 67–68, 2004.

[47] X. Zhu. Semi-supervised learning literature survey, 2006.

[48] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, 2002.

A Additional Figures

Recovered graph signals by Avg_cons, LP_cd, LP_sr and nLasso presented as images for the synthetic grid graph:

Figure A1: Recovered graph signal using Avg_cons algorithm.

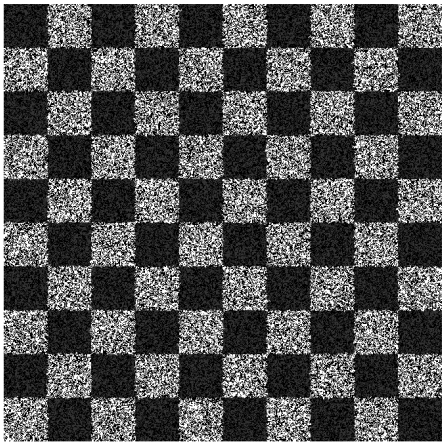

Figure A2: Recovered graph signal using LP_cd algorithm.

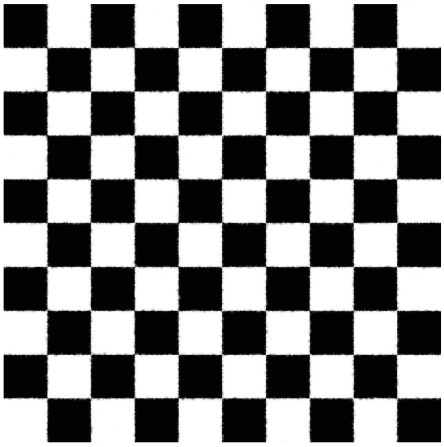

Figure A3: Recovered graph signal using LP_sr algorithm.

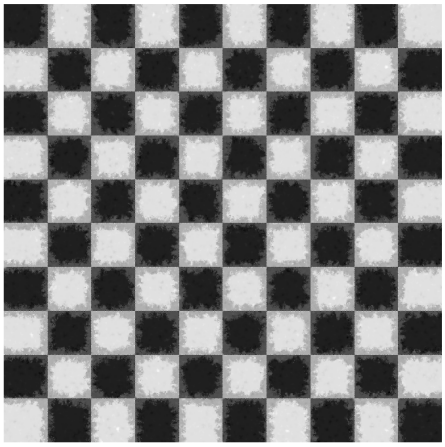

Figure A4: Recovered graph signal using nLasso algorithm.

Recovered graph signals by Avg_cons, LP_cd, SLP and nLasso presented as images for the flower image graph:

Figure A5: Recovered graph signal using Avg_cons algorithm.

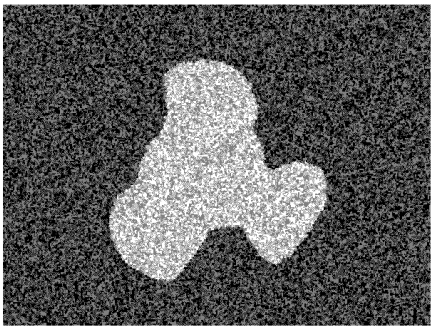

Figure A6: Recovered graph signal using LP_cd algorithm.

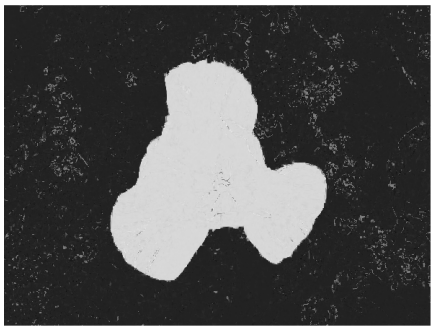

Figure A7: Recovered graph signal using SLP algorithm.

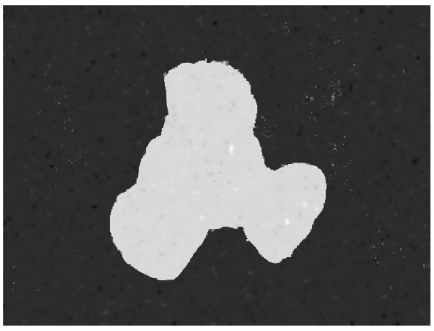

Figure A8: Recovered graph signal using nLasso algorithm.