Statistical and Machine Learning Analysis of the Human Brain Functional Network in a Multi-Site Resting-State Functional MRI Database Framework

Oswaldo Artiles
Florida International University

Fahad Saeed, Ed.
Florida International University, fsaeed@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

STATISTICAL AND MACHINE LEARNING ANALYSIS OF THE HUMAN BRAIN FUNCTIONAL NETWORK IN A MULTI-SITE RESTING-STATE FUNCTIONAL MRI DATABASE FRAMEWORK

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Oswaldo Artiles

2023

To: Interim Dean Ines Triay
College of Engineering and Computing

This dissertation, written by Oswaldo Artiles, and entitled Statistical and Machine Learning Analysis of the Human Brain Functional Network in a Multi-site Resting-state Functional MRI Database Framework, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Date of Defense: November 6, 2023

The dissertation of Oswaldo Artiles is approved.

Florida International University, 2023

DEDICATION

This dissertation is dedicated to Rita Elisa and Ana Rita, my grandmother and my mother, who gave me love and cared for me every day of my childhood, and who taught me the values of honesty, respect for life, and love for hard work, values which have guided me during my entire life, they are always showing and lightning my path.

ACKNOWLEDGMENTS

I would like to acknowledge all the teachers and professors who helped me and taught me during all the years of my life as a student. I am indebted to the members of my dissertation committee: Dr. Janki Bhimani, Dr. Leonardo Bobadilla, Dr. Xudong He, and Dr. Arif Sarwat, for supporting and giving me all their feedback to improve my research. My special acknowledgment to my adviser, Dr. Fahad Saeed, who taught me a lot about computer science and provided me with all the financial and computational resources needed to complete the research presented in this dissertation. I wish to acknowledge the administrative staff of the Knight Foundation School of Computing and Information Sciences (KFSCIS), especially Rebeca, Olga, Luis, Sydney, Eric, and John, who helped me with all their administrative and technical expertise. I also would like to thank Dr. Muhammad Haseeb, Dr. Muhammad Usman Tariq, Fahad Almuqhim, and Tianren Yang, my fellow Ph.D. students at Dr. Saeed's lab, for the very helpful discussions about computer science that I have had with them. Finally, my greatest debt to my wife Ivia, for giving me love and for taking care of me with total dedication, and to my daughter Livia, for all her love and for giving me the most beautiful reason to live.

ABSTRACT OF THE DISSERTATION

STATISTICAL AND MACHINE LEARNING ANALYSIS OF THE HUMAN BRAIN FUNCTIONAL NETWORK IN A MULTI-SITE RESTING-STATE FUNCTIONAL MRI DATABASE FRAMEWORK

by

Oswaldo Artiles

Florida International University, 2023

Miami, Florida

Professor Fahad Saeed, Major Professor

The human brain has a complex network structure that is non-random and multi-scale. It consists of subsystems coupled by a nonlinear dynamic, enabling it to produce complex responses to various external inputs and self-organize. To understand the physical structure and specific brain functions, it is essential to comprehend the connectivity of the hundreds of billions of neurons in the human brain. Functional connectivity (FC) in modern neuroscience is the statistical temporal dependencies between neuronal activation events occurring in spatially separated brain regions. Resting-state functional magnetic resonance imaging (rs-fMRI) is a non-invasive imaging technique widely used in neuroscience to understand the functional connectivity of the human brain. The studies presented in this dissertation were based on the models and methods from network neuroscience, which is an active area of research developed in the last three decades. These methods were used to model and analyze the functional human brain networks in a multi-site rs-fMRI data framework.

The contributions made in this dissertation to the study of the functional connectivity of the human brain network are:

1. The GPU-based Sparse Fast Fourier Transform (SFFT) of k-sparse signals;
2. The GPU-based breadth-first search algorithm;
3. The GPU-based betweenness centrality graph metric algorithm;
4. A comprehensive approach to solving the problem of confounding effects in the machine learning classification models of rs-fMRI multi-site data; and
5. A preliminary assessment of time-varying functional connectivity in a multi-site data rs-fMRI framework.

We hope that the neuroscience research community will use and improve these contributions to enhance the discovery of the functions and structure of the human brain. This will lead to a better understanding of the causes of brain disorders and the development of useful and effective biomarkers for their diagnosis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

BC	Betweenness Centrality
BFS	Breadth-First Search
COO	Coordinate sparse storage format
COOC	Coordinate Column sparse storage format (transpose of the COO format)
CSC	Compressed Sparse Column storage format
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
cuFFT	NVIDIA CUDA Fast Fourier Transform (FFT) library
FD	Framewise displacement
FFT	Fast Fourier Transform
fMRI	Functional Magnetic Resonance Imaging
GLT	Global Memory Load Throughput
GPU	Graphics Processing Unit
GTEPs	(Billions of Transverse Edges by Second)
HPC	High Performance Computing
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MIT-SFFT	MIT Sparse Fast Fourier Transform
MRI	Magnetic Resonance Imaging
MSE	Mean Square Error
MTEPs	(Millions of Transverse Edges by Second)

rs-fMRI	resting state functional Magnetic Resonance Imaging
SFFT	Sparse Fast Fourier Transform
SpMV	Sparse matrix-vector multiplication
tvFC	Time varying functional connectivity

CHAPTER 1

INTRODUCTION

In this chapter, we present a brief review of human brain functional connectivity, as well as an introduction to network and computational neuroscience as a general framework for the studies presented in this dissertation. We also include a brief description of the goals and main results of our contributions to network and computational neuroscience.

1.1 Human Brain Functional Connectivity

The human brain is the most complex physical network which we know. A human brain is composed of nerve cells, or neurons, elements for processing information and signaling, and glial cells which are supporting elements. There are about 95 billion neurons and the same number of nonneuronal cells in the human brain [ACG+09], connected by approximately 100 trillion synapses, connections through which a neuron receives information from other neurons [VG20], a model first proposed in the seminal works of Ramon y Cajal [RyC06, yC97].

The human brain has a structure physically organized over three scales of space: microscopic, mesoscopic, and macroscopic [BWB+09, STK05]. The microscopic scale is represented by individual neurons and synapses ($> 1\mu$ m), this scale refers to properties of the human brain that are only visible with microscopy devices. In this scale, the reconstruction of networks of individual neurons and synapses requires the use of invasive techniques over sections of brain tissues [FZB16]. The mesoscale ($\sim 0.3-3.3$ mm) refers to populations of neurons such as cortical columns, of similar types, sharing similar properties. In this scale, a combination of microscopy and macroscopic techniques are combined to understand neuronal connectivity for a large part of the brain. In the macroscopic scale, the brain is divided into a set of distinct and coherent regions of interest (ROI), where each ROI has a biological meaning [Spo11]. In this scale, noninvasive imaging techniques such as magnetic resonance imaging (MRI) of the entire brain are used to map its structural and functional connectivity on a range of millimeters to centimeters [FZB16, BS05, BWB+09].

Understanding the connectivity of the hundreds of billions of neurons in the human brain is essential to knowing its physical structure and the nature of specific brain functions. In modern neuroscience, there are three important interrelated concepts of connectivity: structural or anatomic, functional, and effective. The *structural connectivity or connectome* [STK05, Hag05] refers to the set of physical connections between neuronal elements forming the human brain, mainly at the macroscopic and mesoscopic scales [STK05, Spo11, CJY+13]. *Functional connectivity (FC)* is defined as the statistical temporal dependencies between neuronal activation events occurring in spatially separated brain regions [FFLF93]. *Effective connectivity (EC)* is described as the influence of a neuronal system on another [FFF93]. A detailed review of functional and effective connectivity is given in reference [Fri11].

Resting-state functional magnetic resonance imaging (rs-fMRI) is a non-invasive imaging technique based on the blood oxygen level of the brain [OLNG90, OMT+93], widely used in neuroscience to understand the functional connectivity of the human brain (see Section 5.2.1). At the macroscopic scale, functional connectivity (FC) can be measured as the statistical correlations between rs-fMRI time series recorded at different brain regions. In imaging neuroscience, there are two classes of functional connectivity: Static functional connectivity and time-varying functional connectivity. Static functional connectivity (sFC) is computed by assuming that functional connectivity is constant in time. sFC is usually computed within the entire rs-fMRI scanning session, capturing instantaneous statistical relationships between brain areas within a scanning session. Time-varying functional connectivity (tvFC) is the functional connectivity that varies as a function of time. tvFC can be computed within given segments of the rs-fMRI scanning session, shorter than the entire scanning session, capturing instantaneous statistical relationships between brain areas within each given segment of the scanning session. The tvFC is therefore a time sequence of sFC values computed from the rs-fMRI time series in each segment. tvFC can be used to extract dynamic functional connectivity from the rs-fMRI data [CG10, HWA$^+$13, ZFC$^+$14, GB14]. In this dissertation, we computed the linear correlation between the time series for all pairs of nodes of the functional networks, using the Pearson correlation function (See Section 5.2.2)

1.2 Network and computational neuroscience: a brief introduction

The human brain has been modeled as a complex network with a non-random multi-scale structure, with subsystems coupled by nonlinear dynamics [BT02], with the capability of generating complex responses to simultaneous and diverse external inputs, and with self-organization capabilities.

Network neuroscience [BS17] is an active area of neuroscience research, developed in the last three decades for the data analysis and modeling of the healthy and diseased functional and structural human brain networks [BS09, BB11, Spo13, Spo11, Spo12, BS17]. Network neuroscience emerges from the intersection of the availability of large and complex neural imaging data recorded from the human brain, and the development of modern network science, a theoretical and practical framework for the study of networks [BSV$^+$07].

Computational neuroscience an important branch of computer science, has been fundamental for the progress of network neuroscience in the last three decades. Computational neuroscience encompasses a myriad of applications ranging from statistical techniques for processing and analysis of brain imaging [Fri07, JBB13], computational modeling of dynamic neural systems [DA05, FD10, Coo10, KAA$^+$18, KD18, KD20, SSZ23, SJD$^+$23], graph theoretical analysis of structural and functional brain networks [BS09, PF13, MWBV18, Spo18], and statistical and machine learning analysis of brain networks for classification and discovery of biomarkers for brain disorders [FZB15, YKK17, MWBV18].

In network neuroscience, the resting-state functional magnetic resonance imaging (rs-fMRI) time series are modeled as a functional brain network, which is represented as a weighted, undirected graph $G = (V, E)$, where V is the finite set of nodes, representing brain regions, and E the set of edges, representing the connections between the brain regions [Spo13]. The weights of the edges of the functional networks are equal to the values of the static functional connectivity between the brain regions (see Section 5.2.2).

The importance of nodes (brain regions) and edges (connections between brain regions) of the functional networks are usually measured using graph metrics such as node strength, closeness centrality, node betweenness centrality (BC), eigenvector centrality (EVC), and clustering coefficient (CC). A node strength is computed as the sum of the weights of the edges attached to the node. The closeness centrality of a node is equal to the inverse of the average shortest path distance to the node. A node betweenness centrality (BC) measures the proportion of shortest paths between all pairs of nodes in the graph that passes through the node. The eigenvector centrality of a node in a graph is a measure of the centrality of its neighbors, which is formally defined as the corresponding element of the leading eigenvector of the adjacency matrix representing the graph. The clustering coefficient (CC) of a node is the proportion of closed triangles that are attached to the node, relative to the total number of closed triangles that are possible between the neighbors of the node. These local topological graph measures have been applied to the detection of hubs, highly and very important brain regions, in the human brain network [SHK07], as well as in the dynamic functional network of the human brain at rest [KEFK^{+}17].

1.3 Contributions to Network and Computational Neuroscience

In this dissertation, we present two groups of contributions to network and computational neuroscience applied to the study of the functional connectivity of the human brain network.

Our first group of contributions is the design and development of GPU-based high-performance algorithms to compute: 1) the Sparse Fast Fourier Transform (SFFT) of k-sparse signals [AS19]; 2) the breadth-first search algorithm [AS21d]; and 3) the betweenness centrality graph metric [AS21c], all of which can be used for the analysis of large structural and functional brain networks.

Our second group of contributions, applicable to the statistical and machine learning analysis of human brain functional networks in a multi-site resting-state functional MRI database framework, are: 1) A comprehensive approach to the solution of the problem of confounding effects over the machine learning classification models of rs-fMRI multi-site data [AS21b, AAMS23], and 2) a preliminary assessment of time-varying functional connectivity in a multi-site data rs-fMRI data framework. In the following sections, we summarize our contributions.

1.3.1 High-performance Algorithms

GPU-SFFT: A GPU Based Algorithm to Compute the Sparse Fast Fourier Transform (SFFT) of k-sparse Signals (Chapter 2)

The Sparse Fast Fourier Transform (MIT-SFFT) is a sequential algorithm developed to compute the discrete Fourier transform of a signal with a sublinear time complexity, i.e., algorithms with runtime complexity proportional to the sparsity level k, where k is the number of non-zero coefficients of the signal in the frequency domain. In this dissertation, we proposed a highly scalable and parallel high-performance CPU-GPU algorithm called GPU-SFFT for computing the SFFT of k-sparse signals.

Our experiments showed that the specific optimizations applied to the design of GPU-SFFT resulted in large decreases in the execution times needed for computing the SFFT. The comparison of the execution times of the GPU-SFFT with the execution times of the sequential MIT-SFFT algorithm showed that the speedup obtained with the GPU-SFFT was up to 38x when the times for data transfer between the CPU and the GPU were not considered, and up to 22x when these times were included. Moreover, up to 5x speedup was obtained when the execution times of the GPU-SFFT were compared to the corresponding times of cuFFT, the NVIDIA CUDA Fast Fourier Transform (FFT) library.

TurboBFS: A GPU Based Breadth-first Search (BFS) Algorithms (Chapter 3)

TurboBFS is a GPU based linear-algebraic formulation and implementation of the well-known Breadth-First Search (BFS) algorithm, that exhibits excellent scalability on unweighted, undirected, or directed sparse graphs of arbitrary structure.

Our experimental results demonstrated that our TurboBFS algorithms obtained up to 40 GTEPs (billions of transverse edges per second), and were on average 15.7x, 5.8x, and 1.8x faster than the other state-of-the-art BFS algorithms implemented on the sequential SuiteSparse:GraphBLAS, and GPU-based GraphBLAST, and gunrock libraries, respectively.

TurboBC: A GPU Based Betweenness Centrality Algorithms (Chapter 4)

TurboBC is, as far as we know, the first GPU-based linear-algebraic formulation and implementation of a set of memory-efficient betweenness centrality algorithms that exhibits good performance and high scalability on unweighted, undirected, or directed sparse graphs of arbitrary structure.

Our extensive set of experiments showed that the TurboBC algorithms obtained more than 18 GTEPs (billions of transverse edges per second), and an average speedup of 31.9x over the sequential version of the BC algorithm, and were on average 1.7x and 2.2x faster than the state-of-the-art BC algorithms implemented on the high performance, GPU-based, gunrock [WDP+16], and CPU-based, ligra [SB13] libraries, respectively. These experiments also showed that by minimizing their memory footprint, the GPU memory usage of of the gunrock library was higher than the memory usage of the TurboBC algorithms, allowing these algorithms to compute the BC of relatively big graphs, for which the gunrock algorithms ran out of memory. Our experiments also demonstrated that the performance obtained by the TurboBC algorithms, measured as MTEPs, as a function of the GPU memory bandwidth, were much greater than those obtained by the BC algorithms in the gunrock library, showing that the GPU memory was used more efficiently by the TurboBC algorithms.

1.3.2 Statistical Analysis of Human Brain Functional Networks

The preprocessed rs-fMRI data used in the studies presented in this Section was obtained from the international imaging sites publicly available in the ABIDE database [CBC+13, DMYL+14, DMOC+17].

A comprehensive approach to the solution of the problem of confounding effects over the machine learning classification models of rs-fMRI multi-site data (Chapter 5)

One main challenge for the neuroscience research community using rs-fMRI multi-site databases is the existence of confounding effects, associated with variables resulting from imaging and population heterogeneity among different sites. Several studies have shown that these confounding factors affect the performance of the machine learning models when executed on rs-fMRI multi-site data [PBM15, KFMB+16, AMDM+17]. One main effect is the increase in variability, as well as the imposition of upper limits on the classification scores, due to the decrease of statistical power of the machine learning classification of patients and control subjects.

As a solution to this challenge, we proposed a comprehensive approach to improving the classification scores of the machine learning models applied to the analysis of multi-site rs-fMRI data by i) identifying the population and imaging variables producing the confounding effects, and ii) controlling these confounding effects to maximize the classification scores. For this study, we computed functional networks derived from static functional connectivity.

The main results obtained with our proposed models and methods were an accuracy of 76.4 %, sensitivity of 82.9 %, and specificity of 77.0 %, which are 8.8 %, 20.5 %, and 7.5 % above the baseline classification scores obtained from the machine learning analysis of the static functional connectivity computed from the ABIDE rs-fMRI multi-site data. These experimental results demonstrated the effectiveness of our proposed approach to quantify the confounding effects of the phenotypic and imaging variables, as well to maximize the classification scores which were obtained with the proposed statistical models and methods.

Assessment of Time-Varying Functional Connectivity in a Multi-site rs-fMRI Data Framework (Chapter 6)

Considering that the human brain is a complex non-linear dynamic system [BT02], the assumption of static functional connectivity is an important limitation to advancing our knowledge of the dynamic functional brain. Considering this limitation, recent research has evolved to use the concept of time-varying functional connectivity (tvFC), i.e., functional connectivity, measured as the statistical correlations between rs-fMRI time series recorded at different brain regions, that vary as a function of time.

In this preliminary study, we used functional networks with the nodes representing brain regions. The rs-fMRI time series of each brain region was segmented using a sliding time-window technique, and then the time-varying functional connectivity (tvFC) was obtained as a time-sequence of static functional connectivity (sFC) values computed for each segment. We performed statistical tests on the tvFC for each ABIDE subject, to prove the claim that the time-variability of each tvFC represents non-linear dynamics of the functional brain.

To prove the null hypothesis H_0, i.e., the claim that the variability of tvFC is due to measurement noise, we needed to obtain the test statistics of the null distribution to verify whether the test statistics of the tvFC, obtained from the rs-fMRI time series, are outside the test statistics of the null distribution. Since the amount of the ABIDE rs-fMRI data is very limited to compute the test statistics of the null distribution, a solution is to generate this distribution from surrogate data [TEL+92].

The goal of this preliminary study was the assessment of the effects of the width in seconds of the time windows, the filtering and global signal regression, and the head motion over the non-linear dynamics of the functional brain network contained in the tvFC values obtained from the ABIDE rs-fMRI data.

Our experimental results showed that the two greatest number of subjects for which the null hypothesis H_0 was rejected, were obtained for the no-filtering strategies (nofilt-global and nofilt-noglobal) of the CPAC pipeline, respectively. We also showed that due to the greater cutoff frequencies corresponding to time windows with a width size of 50 seconds compared to the frequencies obtained for a width size of 100 seconds, the maximum number of subjects rejecting H_0 was 801 subjects for a width size of 50 seconds (91 % of all ABIDE subjects in Table 6.1), 35 % greater than the 594 subjects obtained for a width size of 100 seconds. These maximum results were obtained for the nofilt-global strategy of the CPAC pipeline. Furthermore, these results also allowed us to conclude that for these maximum numbers, the null hypothesis H_0, i.e., the claim that the variability of tvFC is due to measurement noise, was rejected with a P-value of 0.1, and, therefore there is sufficient evidence, at a P-value of 0.1, to support the claim that the variability of tvFC represents non-linear dynamics of the functional connectivity (DFC) of the brain. Finally, we concluded that the effect of the head motion over the non-linear dynamics of the functional brain network contained in the tvFC values had an upper bound determined by the percentages of FD greater than 0.2 mm. This upper bound, for example, was less than 12 % for about 80 % of the subjects rejecting $H0$, hence, for these subjects, a high percentage of the variability of the tvFC values represented the non-linear dynamics of the functional brain network, with the corresponding levels of significance determined by the P-values.

We hope that the contributions of this dissertation will be used and improved by the neuroscience research community, to enhance the discovery of the functions and structure of the human brain, and the understanding of the causes of brain disorders, as well as to define useful and effective biomarkers for the diagnosis of these disorders.

CHAPTER 2

GPU-SFFT:A GPU-BASED ALGORITHM FOR COMPUTING THE SPARSE FAST FOURIER TRANSFORM (SFFT) OF K-SPARSE SIGNALS

In this chapter, we present a highly scalable GPU-based high-performance algorithm, called GPU-SFFT, for computing the Sparse Fast Fourier Transform (SFFT) of k-sparse signals [AS19].

2.1 Introduction

The Discrete Fourier Transform (DFT) is one of the most important algorithms for the analysis of the spectral representation of signals in engineering and science, with a wide range of applications from astronomy to medical imaging, and from seismology to cryptography. The DFT algorithm computes the Fourier transform of a discrete signal of size n with a time complexity of $O(n^2)$. This time complexity results in unacceptable performance for processing the big data sets characteristic of modern applications. The Fast Fourier Transform (FFT) is the fastest and most widely used algorithm to process the DFT of a signal of size n with a time complexity of $O(n\ log\ n)$[DL42, CT65, CLW67, DS00]. However, the FFT algorithm may be too slow to process the DFT of terabyte signals, even when these signals are sparse in the frequency domain, i.e. with only k frequency coefficients different than zero, where $k << n$. Moreover, in many applications, it is sometimes hard to obtain enough data to compute the DFT with the desired accuracy [Has18].

The suboptimal performance of the FFT algorithm to compute the DFT of k-sparse signals, and the existence of an important set of domains(video, audio, medical imaging, spectroscopy, GPS, seismic data) with signals that are sparse in the frequency domain, motivated the development of sub-linear algorithms, i.e. algorithms with runtime complexity proportional to the level of sparsity, which use only a subset of this data to compute the frequency coefficients which are significant [GGI$^+$02, GMS05, GI10]. The Sparse Fast Fourier Transform (SFFT) algorithm developed at MIT (MIT-SFFT) [Has18] are sub-linear algorithms that by exploiting the inherent sparsity of natural signals, are faster than the FFT algorithms for k-sparse data. The MIT-SFFT algorithm implements a solution to the problem of computing the DFT, $\hat{x}(f)$, of a signal, $x(t)$, of size n, with only k non-zero frequency coefficients, the other $n-k$ coefficients are zero. For general signals, the MIT-SFFT computes an approximation $\hat{x}'(f)$ of $\hat{x}(f)$. The time complexity of the version 2.0 of the MIT-SFFT algorithm is $O(\log\ n\sqrt[3]{nk^2 \log n})$ with a sparsity level range of $O(n/\sqrt{\log}\ n)$, i.e. the algorithm is faster than FFT up to $O(n/log\ n)$ [Has18].

Our GPU-based Sparse Fast Fourier Transform (GPU-SFFT) is a GPU-based high-performance parallel algorithm that implements the parallel version of the functionality corresponding to the stages shown in Figure 2.1 for the MIT-SFFT sequential algorithm. GPU-SFFT algorithm is based on carefully designed parallel computing optimization that considers the structure of the MIT-SFFT algorithm as well as CPU-GPU architectures.

The first optimization was to unroll the for loops in the sequential MIT-SSFT to increase the parallelism by maximizing the number of concurrent threads executing independent instructions, as well as ensuring coalesced global memory access by the threads in each warp. The second one was to minimize the transfer of data between the CPU and the GPU, by transferring only input data from the CPU to the GPU, and only output results from the GPU to the CPU, and performing all the computations on the GPU side. The third optimization was to replace the sequential sort algorithms in the MIT-SFFT with the high-performance sorting algorithms available in the Thrust library for CUDA [BH12], as well as to compute the reduced size FFTs of the algorithm with cuFFT, the NVIDIA CUDA Fast Fourier Transform (FFT) library [NVI].

2.2 Computational stages of the MIT-SFFT algorithm

Figure 2.1 shows a simplified workflow diagram of the the computational stages of the sequential MIT-SFFT algorithm [Has18]. The *outer loop* of the algorithm is divided into *location loops* and *estimation loops*.

The location loops in Figure 2.1 implement the stages of *permutation and filtering*, *FFT and cutoff*, and *reverse hash function*. The MIT-SFFT algorithm is based on binning the large Fourier coefficients of the significant frequencies, present in the input signal, into a small set of B bins, where $B = O(k)$ is a parameter that divides n.

The collision problem of having non-zero frequencies coefficients in the same bin is solved by separating two coefficients that are close to each other, locating them on separate bins, so that there is only one large frequency per bin with high probability. The permutation stage of the location loop performs the random permutation of the input signal so that adjacent coefficients in the frequency domain are evenly separated. The process of binning the frequency coefficients is implemented by filtering the permuted input signal. The filter suppresses the frequency coefficients that hash out of the bin while passing through frequency coefficients that hash into the bin. The utilization of a filter guarantees the goal of binning one frequency coefficient per bin, minimizing the leakage of frequencies from one bin to another [Has18]. The hashing-based spectrum permutation method implemented in the

MIT-SFFT maps indices of the original signal spectrum to the permuted locations so that the original locations can then be recovered in the estimation loops.

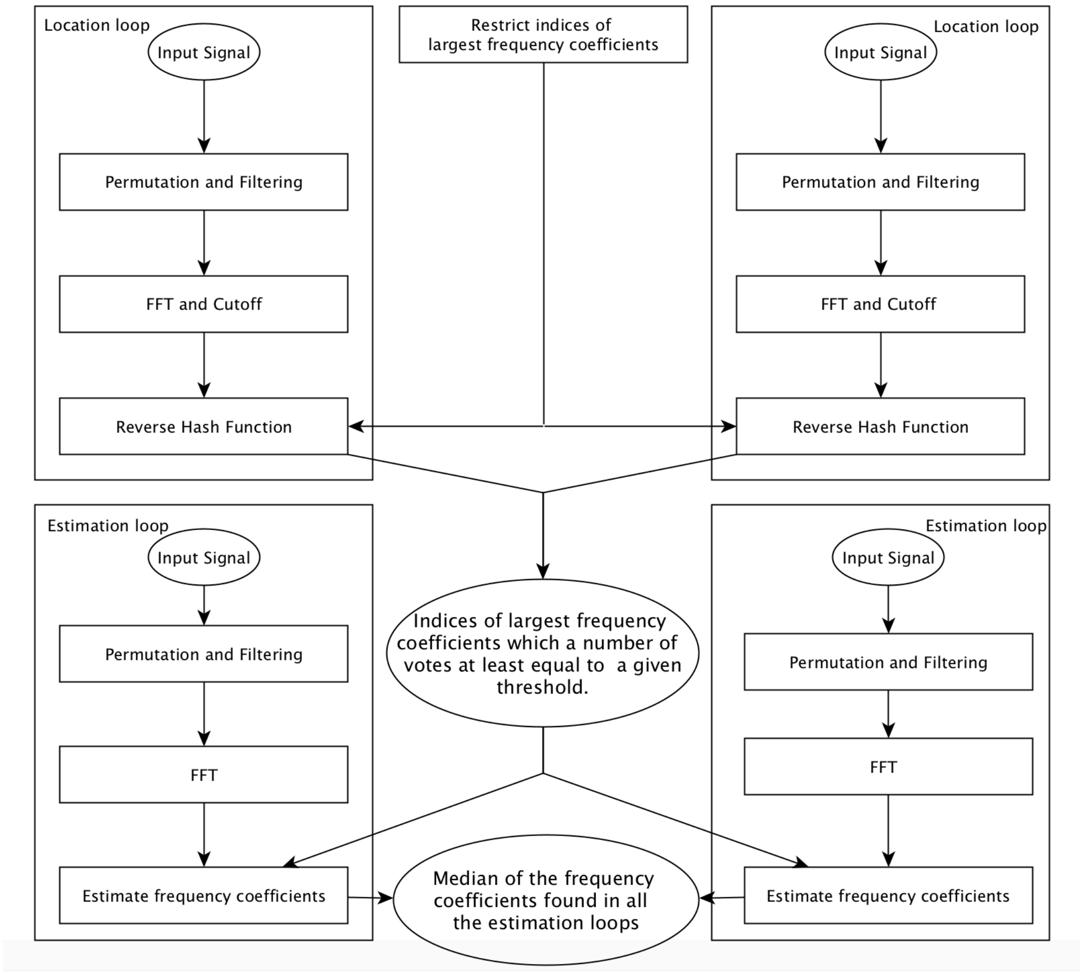

Location loop
Input Signal
Permutation and Filtering
FFT and Cutoff
Reverse Hash Function
Restrict indices of largest frequency coefficients
Location loop
Input Signal
Permutation and Filtering
FFT and Cutoff
Reverse Hash Function
Estimation loop
Input Signal
Permutation and Filtering
FFT
Estimate frequency coefficients
Indices of largest frequency coefficients which a number of votes at least equal to a given threshold.
Median of the frequency coefficients found in all the estimation loops
Estimation loop
Input Signal
Permutation and Filtering
FFT
Estimate frequency coefficients

Figure 2.1: GPU-SFFT: MIT-SFFT Algorithm Workflow
A simplified workflow diagram of the outer loop of the version 2.0 of the MIT-SFFT algorithm

After permuting and filtering the input signal, the original problem has been reduced from a n-dimensional DFT to a B-dimensional DFT. This size-reduced DFT is computed by the FFT algorithm with time complexity $O(B \log B)$. After computing the B-dimensional DFT, each bin in the frequency domain contains at most one potential large coefficient. However, in a k-sparse DFT, it is likely that many of the coefficients in the bins are close to zero. The algorithm guarantees that the probability of missing a large coefficient is low if the top $O(k)$ samples are selected [Has18]. Then, in the cutoff stage, the indices of the top k coefficients of maximum magnitude are selected from the set of B bins in the frequency domain, and the indices of the other $B - k$ coefficients are discarded.

In the reverse hash function stage, the k largest coefficients found in the cutoff stage are reconstructed by finding the original locations in the frequency domain [SP14]. The version 2.0 of the MIT-SFFT adds a heuristic to restrict the location of the indices of the largest coefficients using a filter that has no leakage at all. The intersection of these indices with the indices of the k largest coefficients found in the cutoff stage is computed in the reverse hash function. In a voting approach, this function gives a vote to an index, every time the index appears in the intersection of both sets of indices, when the number of votes reaches a threshold, the index is added to the output of the function. The output of the location loops are the indices of the largest frequency coefficients that have a number of votes at least equal to a given threshold [Has18].

The estimation loops share the permutation, filtering, and the FFT stages described for the location loops, with the goal of having one large frequency coefficient per bin with high probability. Given the set of indices computed by the location loops, and the bins in the frequency domain, the *estimation coefficients* stage of the estimation loops returns the indices and the values of the k largest frequency coefficients. To compensate for the collision of large frequencies in the same bin, the output of the estimation loops is the median of the values of the frequency coefficients found on all the estimation loops.

2.3 GPU based Sparse Fast Fourier Transform algorithm

This section includes a description of the functionality of our GPU-SSFT algorithm, including the techniques used to implement such functionality. The description is based on pseudocodes that map to the stages shown in Figure 2.1. One of the goals in the design of our GPU-SFFT algorithm was to minimize the transfer of data between the CPU and the GPU to reduce the I/O time and to improve speedups. The first procedure to achieve this goal is to compute the time and frequency components of the filters on the GPU (device) side as a preprocessing stage for the GPU-SFFT algorithm. The second procedure is to transfer the data of the CPU (host) input signal of size n to the GPU (device) side only one time at the start of the computation. Both procedures are represented in Algorithm 2.1 for the outer loop function of the GPU-SFFT. The input of Algorithm 2.1 are the host input signal, hx, and the time and frequency components on the device side of the filters, $dfilt_t$ and $dfilt_f$ respectively, where fs is the size of the filters. Lines 4 and 5 of Algorithm 2.1, show the allocation of GPU global memory for the device input signal, dx, and the memory transfer of hx to dx. Lines 6 to 10 of Algorithm 2.1, show the allocation of GPU global memory for the internal variables. The functions which are called in the for loops of Algorithm 2.1 implement the GPU version of the stages shown in Figure 2.1. The output of Algorithm 2.1 is the output signal, $\hat{h}x$, in the device side. All the functions included in Algorithm 2.1 are described in the following sections.

2.3.1 GPU-SFFT: Permutation and filtering

Algorithm 2.2 is the sequential version of the permutation and filtering stage in Figure 2.1, implementing (line 6) the hash function given in Equation 2.1 which

Algorithm 2.1 GPU outer loop function.

```
1: Input: $hx[0..n]$,$dfilt_t[0..fs]$,$dfilt_f[0..fs]$
2: Output: $\hat{hx}[0..IF]$
3: procedure OutLpGPU($hx$,$n$,$dfilt_t$,$dfilt_f$,$fs$,$B$,$B_t$,$W$,$L$,$L_c$,$L_t$,$L_l$)
4:     $dx[0..n] \leftarrow$ cudaMalloc($n$)
5:     cudaMemcpy($dx, hx$)
6:     $dbins_t[0..L*B] \leftarrow$ cudaMalloc($L*B$)
7:     $dbins_f[0..L*B] \leftarrow$ cudaMalloc($L*B$)
8:     $dI[0..n] \leftarrow$ cudaMalloc($n$)
9:     $dJ_2[0..L_c*B_t] \leftarrow$ cudaMalloc($L_c*B_t$)
10:    $dH_\sigma[0..L] \leftarrow$ cudaMallocManaged($L$)
11:    for $i \leftarrow 0, L_c$ do
12:        LocLargeCoefGPU($dx$,$B_t$,$n$,$W$,$dJ_2[i*B_t]$)
13:    end for
14:    $IF \leftarrow 0$
15:    for $i \leftarrow 0, L$ do
16:        $\sigma \leftarrow random()\ mod\ n$                      $\triangleright\ gcd(\sigma, n) = 1$
17:        $dH_\sigma[i] \leftarrow$ modInv($\sigma, n$)             $\triangleright\ dH_\sigma[i]\sigma = 1(mod\ n)$
18:        $dJ[0..B_t] \leftarrow$ cudaMalloc($B_t$)
19:        PermFilterGPU($dx$,$B$,$dfilt_t$,$fs$,$dbins_t$,$dH_\sigma[i]$)
20:        FFTCutoffGPU($dJ$,$dbins_t$,$dbins_f$,$B$,$B_t$)
21:        if $L < L_l$ then
22:            RevHashGPU($dI$,$dJ$,$B_t$,$B$,$n$,$L_t$,$dJ_2$,$W$,$IF$,$\sigma$)
23:        end if
24:    end for
25:    $\hat{hx} \leftarrow$ EstValGPU($dI, IF, dbins_f, dfilt_f, B, n, L, dH_\sigma$)
26: end procedure
```

maps each one of the n elements of the input signal to one of the B bins[Has18].

$$h_{\sigma,B}(i) = floor\left(\frac{i\sigma}{n/B}\right) \quad \triangleright\ h_{\sigma,B} : [n] \rightarrow [B], \tag{2.1}$$

On the GPU version of Algorithm 2.2, a thread collision can occur when multiple threads, for example, threads $i, B+i, 2B+i, ...$, try to update the same bin concurrently, introducing time delays that negatively impact the performance of the parallel algorithm. Algorithm 2.3 is the GPU version of the sequential Algorithm 2.2. Algorithm 2.3 is designed to solve the thread collision with a tiling based approach [KWM16]. The number of colliding threads is approximately equal to $T = floor(fs/B)$, where fs is the size of the filter. Since fs is not divisible by B,

Algorithm 2.2 Sequential function to permute and filter the input signal x.

```
1: procedure PermFilter($x$,$B$,$filt_t$,$fs$,$bins_t$,$H_{\sigma,i}$)
2:     for $i \leftarrow 0, fs$ do
3:     end for
4: end procedure
```

when the filter vector is partitioned into T tiles of size B, there are $R = fs \bmod B$ remaining elements of the filter after the T tile. The tiling approach resolves the collision problem because each GPU thread computes T unique components of the permuted and filtered signal and bins them into only one component of the $bins_t$ vector. Each one of the T components is computed by the GPU thread by convoluting one component of the permuted input signal with a filter component that is for each iteration in a different tile and it is unique for each GPU thread. After the $i = (B-1) * T$ iteration of the for loop finishes, the remaining R iterations, on which $i = B$, access the last R elements of the filter.

The kernel PFTKERN of the algorithm 2.3 implements the tilling approach for unrolling the for loop of the sequential algorithm 2.2. Each GPU thread on this kernel bins the components of the permuted and filtered input signal that correspond to one bin, independently of each other thread on the kernel, making this computation free of collisions. Each thread computes the first for loop (lines 17 to 22) of size T. After this for loop finishes, the remaining R components of the filter are convoluted with the permuted input signal in the second for loop (lines 24-28) of PFTKERN. In the sequential algorithm 2.2, the permutation *index* (line 7) has an implicit dependence on the index i of the for loop. In the GPU version, the computation of the indices of the permuted input signal, the filter, and the bins on PFTKERN are explicitly dependent on the index of the thread. The experimental results showed that for input signals with sizes $n < 2^{27}$, the performance of PFTKERN is much higher than the performance of the corresponding for loop on the sequential version. The

Algorithm 2.3 GPU function to permute and filter the input signal x.

```
1: **Input**: $dx[0..n]$,$dfilt_t[0..fs]$
2: **Output**: $dbins_t[0..B]$
3: procedure PMFILTERGPU($dx$,$B$,$dfilt_t$,$fs$,$dbins_t$,$dH_{\sigma,i}$)
4:     $dbins_t \leftarrow$ CUDAMEMSET($dbins_t, 0, B$)
5:     $T \leftarrow fs/B, R \leftarrow fs \ mod \ B$
6:     if $n < 2^{27}$ then
7:         PFTKERN($dbins_t$,$dx$,$dfilt_t$,$n$,$B$,$dH_{\sigma,i}$,$T$,$R$)
8:     else
9:         PFKERN($dbins_t$,$dx$,$dfilt_t$,$n$,$B$,$dH_{\sigma,i}$,$fs$)
10:    end if
11: end procedure

12: procedure PFTKERN($dbins_t$,$dx$,$dfilt_t$,$n$,$B$,$dH_{\sigma,i}$,$T$,$R$)
13:    $i \leftarrow$ threadIdx.x + blockIdx.x * blockDim.x
14:    if $i < B \ or \ i == B$ then
15:        if $i < B$ then
16:            for $j \leftarrow 0, T$ do
17:                $id \leftarrow i + j * B$
18:                $dbins_t[i] + \leftarrow dx[(id * dH_{\sigma,i}) \ mod \ n] * dfilt_t[id]$
19:            end for
20:        end if
21:        if $i == B$ then
22:            for $j \leftarrow 0, R$ do
23:                $id \leftarrow j * T + i$
24:                $dbins_t[j] + \leftarrow dx[(id * dH_{\sigma,i}) \ mod \ n] * dfilt_t[id]$
25:            end for
26:        end if
27:    end if
28: end procedure

29: procedure PFKERN($dbins_t$,$dx$,$dfilt_t$,$n$,$B$,$dH_{\sigma,i}$,$fs$)
30:    $i \leftarrow$ threadIdx.x + blockIdx.x * blockDim.x
31:    if $i < fs$ then
32:        $dbins_t[i \ mod \ B] + \leftarrow dx[(i * dH_{\sigma,i}) \ mod \ n] * dfilt_t[i]$
33:    end if
34: end procedure
```

sizes of the for loops on this kernel are $T < 30$ and $R < 1000$, which are efficiently computed by one thread within times that are smaller than the time delays caused by the potential thread collisions. However, for input signals with sizes $n \geq 2^{27}$, the value of R is in the range $[2000, 250000]$, increasing the time to compute the second for loop on PFTKERN, degrading the performance of the algorithm. In order to obtain the expected high performance, the kernel PFKERN (lines 32 to 38) was added to Algorithm 2.3. This kernel is designed to implement a direct unrolling of the corresponding sequential for loop. PFKERN has therefore the time delays introduced by the collision of about T threads trying to update the same bin but does not have the time delays caused by the computation of the second for loop on PFTKERN, which are greater than the corresponding collision times. PFKERN has therefore a higher performance than PFTKERN for input signals with sizes $n \geq 2^{27}$. Hence, in order to guarantee the high performance of Algorithm 2.2 for all the input signals, PFTKERN is selected for input signals with sizes $n < 2^{27}$, and PFKERN for sizes $n \geq 2^{27}$ (lines 7-10).

2.3.2 GPU-SFFT: FFT and cutoff

Algorithms 2.4 and 2.5 are the GPU implementation of the FFT and Cutoff stages on Figure 2.1. In Algorithm 2.4, the FFT of the bins vector in the domain time is computed using the cuFFT library. The bins vector in the frequency domain is

Algorithm 2.4 GPU function to compute the FFT of the bins vector in the time domain, and to find and sort the $B_t = 2k$ indices of the largest frequency coefficients in the bins vector in the frequency domain.

```
1: Input: $dbins_t[0..B]$
2: Output: $dbins_f[0..B]$, $dJ[0..B_t]$
3: procedure FFTCutoffGPU($dJ$,$dbins_t$,$dbins_f$,$B$,$B_t$)
4:     $dbins_f \leftarrow$ cuFFT($dbins_t, B$)
5:     $dJ \leftarrow$ CutoffGpu($dJ, dbins_f, B_t, B$)
6: end procedure
```

the input to Algorithm 2.5, on which the $B_t = 2k$ indices of the largest frequency coefficients on this vector are computed and sorted. Algorithm 2.5 allocates device memory for two vectors: $dsamples_s$ and $dsamples_I$. The components of both

Algorithm 2.5 GPU function to find and sort the $B_t = 2k$ indices of the largest frequency coefficients in the input vector.

```
 1: Input: $d\hat{y}[0..m]$
 2: Output: $dId[0..B_t]$
 3: procedure CutoffGPU($dId$,$B_t$,$d\hat{y}$,$m$)
 4:     $dsamples_s[0..m] \leftarrow$ cudaMalloc($m$)
 5:     $dsamples_I[0..m] \leftarrow$ cudaMalloc($m$)
 6:     SKern($d\hat{y}, dsamples_s, dsamples_I, m$)
 7:     thrust::sort($dsamples_s$)
 8:     $cutoff \leftarrow dsamples_s[m - B_t - 1]$
 9:     $id \leftarrow 0$
10:     CKern($dId, cutoff, dsamples_I, m, id, B_t$)
11:     thrust::sort($dId$)
12: end procedure

13: procedure SKern($d\hat{y}$,$dsamples_s$,$dsamples_I$,$m$)
14:     $i \leftarrow$ threadIdx.x + blockIdx.x $*$ blockDim.x
15:     if $i < m$ then
16:         $dsamples_s[i] \leftarrow ||d\hat{y}||^2$
17:         $dsamples_I[i] \leftarrow dsamples_s[i]$
18:     end if
19: end procedure

20: procedure CKern($dId$,$cutoff$,$dsamples_I$,$m$,$id$, $B_t$)
21:     $i \leftarrow$ threadIdx.x + blockIdx.x $*$ blockDim.x
22:     if $i < m$ then
23:         if $dsamples_I[i] > cutoff$ and $id < B_t$ then
24:             $dId$ [atomicAdd($id, 1$)] $\leftarrow i$
25:         end if
26:     end if
27: end procedure
```

vectors are computed as the square of the magnitudes of the input vector on the SKERN kernel. After sorting the $dsamples_s$ vector, with the sort functionality of the Thrust library [BH12], a cutoff value is computed. The kernel CKERN implements the unrolling of the corresponding for loop on the sequential version of the MIT-SFFT algorithm. The cutoff value is used on CKERN to compute the indices of the largest frequency coefficients. These largest frequency coefficients are the elements in $dsamples_I$ with values greater or equal to the cutoff.

Algorithm 2.6 GPU function to restrict the location of the $2k$ largest frequency coefficients in the DFT of the input signal x.

```
1: Input: $dx[0..n]$
2: Output: $dJ_2[0..B_t]$
3: procedure LocLargeCoefGPU($dx$,$B_t$,$n$,$W$,$dJ_2$)
4:     $dx'[0..W] \leftarrow$ cudaMalloc($W$)
5:     $d\hat{y}[0..W] \leftarrow$ cudaMalloc($W$)
6:     $\sigma \leftarrow n/W, \tau \leftarrow random() * \sigma$
7:     LLCKern($dx', dx, W, \tau, \sigma$)
8:     $d\hat{y} \leftarrow$ cuFFT($dx', W$)
9:     $dJ_2 \leftarrow$ CutoffGPU($dJ_2, B_t, d\hat{y}, W$)
10: end procedure

11: procedure LLCKern($dx'$,$dx$,$W$,$\tau$,$\sigma$)
12:     $i \leftarrow$ threadIdx.x + blockIdx.x $*$ blockDim.x
13:     if $i < W$ then
14:         $dx'[i] \leftarrow dx[\tau + i * \sigma]$
15:     end if
16: end procedure
```

2.3.3 GPU-SFFT: Frequency coefficients of the input signal

The function to restrict the location of the $2k$ largest frequency coefficients in the DFT of the input signal x is a heuristic that was added as a preprocessing stage in version 2.0 of the MIT-SFFT (see Figure 2.1), to improve the performance of the algorithm [Has18]. This function implements an aliasing filter which is very efficient because has no leakage. Algorithm 2.6 implements the GPU version of this function. The output of the kernel LLCKERN is the filtered input signal dx'. After computing the FFT of dx',$d\hat{y}$, by cuFFT, the set of indices of the $2k$ largest frequencies contained in $d\hat{y}$ is computed by the procedure CUTOOFGPU on Algorithm 2.5.

2.3.4 GPU-SFFT: Reverse hash function

The GPU version of the reverse hash stage of Figure 2.1 is implemented by Algorithm 2.7 by unrolling the corresponding for loop on the sequential version of the MIT-

SFFT algorithm. The goal of Algorithm 2.7 is to reverse the hash function (Equation 2.1) to compute the true indices of the largest frequency coefficients that have been hashed to non-empty bins [Has18]. The input of Algorithm 2.7 is the set of indices of frequency coefficients, dJ, computed by Algorithm 2.5, and the set of indices of frequency coefficients, dJ_2, computed by Algorithm 2.6. For each location loop, Algorithm 2.7 computes the set of indices I_L of the largest frequency coefficients that map to J under the hash function and that are in the permuted set of indices $dJ_{2\sigma}$, namely

$$I_L = \{i_L \in [n] | (h_\sigma(i_L) \in dJ) \cap (i_L \in dJ_{2\sigma})\}, \tag{2.2}$$

where $dJ_{2\sigma} = (dJ_2 * \sigma) \bmod W$. Algorithm 2.7 implements a voting approach [Has18] on which each time an index is added to the set I_L, the index will get a vote (line 14). The output of Algorithm 2.7 is the set of true indices dI of the largest frequency coefficients which get a number of votes equal or greater than a given parameter L_t, i.e.

$$dI = \{i_d \in I_j | dV[i_d] \geq L_t\}, \tag{2.3}$$

where $I_j \in I_L$.

2.3.5 GPU-SFFT: Estimate values

The GPU version of the estimate frequency coefficients stage of Figure 2.1 is implemented by Algorithm 2.8 by unrolling the corresponding for loop on the sequential version of the MIT-SFFT algorithm. This algorithm is based on the theorem that the DFT of a signal in the time domain corresponds to phase rotation in the frequency domain, i.e. $x(n-\tau) \Leftrightarrow e^{-2\pi f \tau}\hat{x}(f)$ [SI06]. The goal of Algorithm 2.8 is to compute the true values of the largest frequency coefficients that have been hashed

Algorithm 2.7 GPU function to reverse the hash function and to return the set of indices of the largest frequency coefficients that occurred in at least L_t of the location loops.

```
1: Input: $dJ[0..B]$,$dJ_2[0..B_t]$
2: Output: $dI[0..IF]$
3: procedure RevHashGPU($dI$,$dJ$,$B_t$,$B$,$n$,$L_t$,$dJ_2$,$W$,$IF$,$\sigma$)
4:     $dV[0..n] \leftarrow$ cudaMalloc($n$)
5:     $dV \leftarrow$ cudaMemset($dV, 0, n$)
6:     $dJ_{2\sigma}[0..B_t] \leftarrow$ cudaMalloc($B_t$)
7:     $dJ_{2\sigma} \leftarrow (dJ_2 * \sigma)\ mod\ W$
8:     RHKern($dI$,$dJ$,$dV$,$dJ_{2\sigma}$,$L_t$,$IF$,$W$)
9: end procedure

10: procedure RHKern($dI$,$dJ$,$dV$,$dJ_{2\sigma}$,$L_t$,$IF$,$W$)
11:     $i \leftarrow$ threadIdx.x + blockIdx.x $*$ blockDim.x
12:     if $i < B_t$ then
13:         $I_L = \{i_L \in [n] | (h_\sigma(i_L) \in dJ) \cap (i_L \in dJ_{2\sigma})\} \leftarrow i_{L,i}$
14:         atomicAdd($dV[i_{L,i}], 1$)
15:         if $dV[i_{L,i}] == L_t$ then
16:             $dI$ [atomicAdd($IF, 1$)] $\leftarrow i_{L,i}$
17:         end if
18:     end if
19: end procedure
```

to non-empty bins, by removing the phase rotation introduced by the permutation and filtering of the input signal in the time domain [Has18]. The input of Algorithm 2.8 is the set of indices of frequency coefficients, dI, computed by Algorithm 2.7. Each thread of the kernel EVKERN of Algorithm 2.8 runs a for loop of size equal to the total number of loops (L), on which the index $I(i)$ is permuted, and the value of the largest frequency coefficient $bins_f[h_{\sigma,B}(I[i])]$ is divided by the corresponding component of the frequency component of the filter to remove the phase rotation. It is possible that more than one frequency hash to the same bin, hence, to compensate for errors due to this hash collision, the median of the values computed in the for loop is assigned to the $I(i)$ component of the $SFFT(x) = \hat{x}$. The device memory transfer of the SFFT output signal, $d\hat{x}$ to the host memory, $h\hat{x}$, completes the output of Algorithm 2.8.

Algorithm 2.8 GPU function to estimate the values of the largest coefficients given the indices of such coefficients.

```
1: Input: $dI[0..IF]$,$dfilt_f[0..fs]$,$dbins_f[0..L*B]$
2: Output: $h\hat{x}[0..IF]$
3: procedure EValGPU($dI$,$IF$,$dbins_f$,$dfilt_f$,$B$,$n$,$L$,$dH_\sigma$)
4:     $d\hat{x}[0..IF] \leftarrow$ cudaMalloc($IF$)
5:     EVKern($d\hat{x}$,$dI$,$IF$,$dbins_f$,$L$,$n$,$dH_\sigma$,$B$,$dfilt_f$)
6:     cudaMemcpy($h\hat{x}$, $d\hat{x}$)
7:     return $h\hat{x}$
8: end procedure

9: procedure EVKern($d\hat{x}$,$dI$,$IF$,$dbins_f$,$L$,$n$,$dH_\sigma$,$B$,$dfilt_f$)
10:     $i \leftarrow$ threadIdx.x + blockIdx.x $*$ blockDim.x
11:     if $i < IF$ then
12:         $pos \leftarrow 0$
13:         for $j \leftarrow 0, L$ do
14:             $id \leftarrow (dH_\sigma[j] * dI[i]) \; mod \; n$
15:             $x'_v[pos] \leftarrow bins_f[h_{\sigma,B}(I[i])]/filt_f[id \; mod \; (n/B)]$
16:             $pos_{++}$
17:         end for
18:         $d\hat{x}[I[i]] \leftarrow median(x'_v)$
19:     end if
20: end procedure
```

2.4 Results

In this Section, we present the results of the experiments performed to compare the performance of the proposed GPU-SFFT algorithm with the performances of the sequential MIT-SFFT algorithm [Has18], and of cuFFT, the NVIDIA CUDA Fast Fourier Transform (FFT) library [NVI]. We were not able to compare the performance of GPU-SFFT with other similar algorithms proposed on reference [WCC16] because neither the code nor the parameters used in their experiments were available.

As a preliminary stage for the design and implementation of GPU-SFFT, we port the MIT-SFFT in C++ to C. This version called MIT-SFFTC is more compatible with the CUDA C language easing the implementation of GPU-SFFT. We used this version for all the experiments included in this Section.

2.4.1 Experiments

The input signal, x, to the experiments in the time domain was computed as inverse DFTs of a signal, $\hat{y}$, in the frequency domain with k randomly chosen elements equal to 1 and $n-k$ elements equal to zero. The output SFFT signals of the experiments, $\hat{x}$, were compared to the corresponding input signal in the frequency domain. Only the experiments whose results replicated exactly the input signals, i.e., with no missing components, were accepted as valid results. For all the experiments presented in this section, the output signal error was measured by the mean square error (MSE) defined by [TD00]

$$MSE = \frac{1}{k} \sum_{i=0}^{k-1} [(\hat{x}[i].x - \hat{y}[i].x)^2 + (\hat{x}[i].y - \hat{y}[i].y)^2], \tag{2.4}$$

where $\hat{x}[i].x(y[i].x)$ and $\hat{x}[i].y(y[i].y)$ are the real and imaginary components of $\hat{x}[i](y[i])$ respectively.

The experiments designed to evaluate the performance of the GPU-SFFT algorithm were divided into two sets. The first set of experiments had input signals with sizes in the range $[2^{19}, 2^{27}]$, and a level of sparsity of $k = 1000$. The second set of experiments had input signals with sparsity levels in the range $[1000, 43000]$, and a size of $n = 2^{27}$. The parameters used in the experiments are given in Tables 2.1 to 2.4, and the symbols used in these tables to identify the parameters are described in the Sparse Fourier Transform Code Documentation [HIKP12].

Table 2.1: Parameters MIT-SFFT ($n = 2^q, k = 1000$)

q	19	20	21	22	23	24	25	26
B	3.85	1.4	2.16	0.683	0.99	0.663	0.662	0.478
Comb-cst	256	128	128	256	256	128	128	128
loc-loops	3	2	2	2	2	2	2	2
est-loops	5	6	5	5	3	5	5	5
thre-loops	2	2	2	2	2	2	2	2
Comb-loops	2	2	2	2	2	2	2	2

Table 2.2: Parameters GPU-SFFT ($n = 2^q, k = 1000$)

q	19	20	21	22	23	24	25	26
B	3.85	1.4	2.16	0.683	0.99	0.663	0.662	0.478
Comb-cst	256	128	128	256	256	128	128	128
loc-loops	3	2	2	2	2	2	2	2
est-loops	5	5	5	5	3	5	5	5
thre-loops	2	2	2	2	2	2	2	2
Comb-loops	2	2	2	2	2	2	2	2

Table 2.3: Parameters MIT-SFFT ($n = 2^{27}$)

k/1000	1	7	13	19	25	31	37	43
B	0.68	0.77	0.665	0.66	0.79	0.69	0.70	0.8
Comb-cst	128	4096	4096	4096	8192	16384	32768	32768
loc-loops	3	2	2	2	2	3	3	3
est-loops	4	5	9	9	9	9	10	9
thre-loops	2	2	2	2	2	2	2	2
Comb-loops	2	2	2	2	2	2	2	2

All the experiments presented in this Section were performed on a Linux server with Ubuntu operating system version 16.04.5, 44 Intel Xeon Gold processors, a clock speed of 2.1 GHz, and 125 GB of RAM. The GPU in this server is an NVIDIA Titan Xp, with 30 SM, 128 cores/SM, maximum clock rate of 1.58 GHz, 12196 MB of global memory, and CUDA version 10.1 with CUDA capability of 6.1.

2.4.2 Experimental Results

For the first set of experiments, Figure 2.2 a) compares the execution times of the MIT-SFFTC and the GPU-SFFT, when the I/O times to transfer the input signal from the host to the device were not included. The GPU-SFFT times reflected the impact of the parallelization by being nearly independent of the signal size and by being much lower than the times of MIT-SFFTC. The speedup obtained with the GPU-SFFT vs MIT-SFFTC (Figure 2.4 a) had a maximum of 21x with an average of 9x. When the I/O times were included, GPU-SFFT was faster than MIT-SFFTC for all signal sizes, and faster than cuFFT for signal sizes greater than 2^{22} (Figure

Table 2.4: Parameters GPU-SFFT ($n = 2^{27}$)

k/1000	1	7	13	19	25	31	37	43
B	0.68	0.77	0.665	0.66	0.79	0.69	0.70	0.8
Comb-cst	128	4096	4096	4096	8192	16384	32768	32768
loc-loops	3	2	2	2	2	3	3	3
est-loops	4	5	7	9	9	9	8	8
thre-loops	2	2	2	2	2	2	2	2
Comb-loops	2	2	2	2	2	2	2	2

2.2 b)). For this case, the speedup obtained with GPU-SFFT vs MIT-SFFTC had a maximum of 4x and an average of 3x, and the speedup of GPU-SFFT vs cuFFT had a maximum of 5x and an average of 3x.

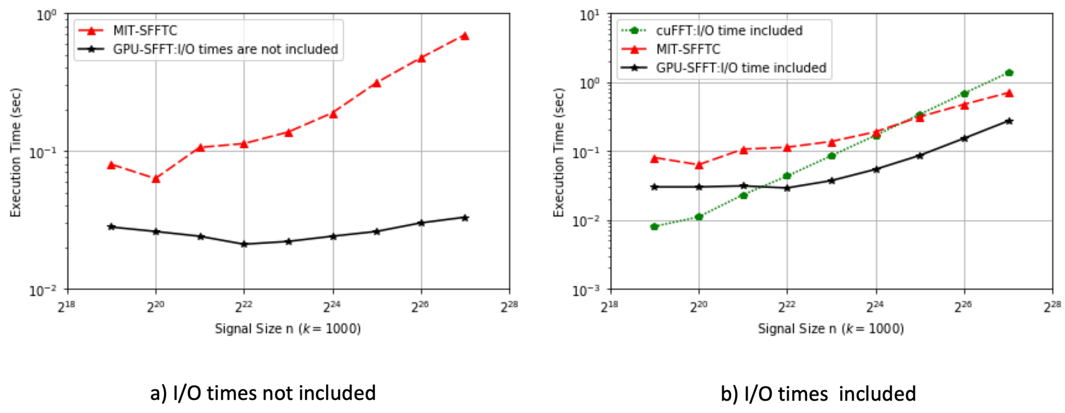


Figure 2.2: GPU-SFFT: Execution Times vs Signal Size

For the second set of experiments, Figure 2.3 a) compares the execution times of the MIT-SFFTC and the GPU-SFFT when the I/O times are not included. The GPU-SFFT times were much lower than the times of MIT-SFFTC in the complete range of sparsity levels. The speedup obtained with the GPU-SFFT vs MIT-SFFTC (Figure 2.4 b) had a maximum of 38x with an average of 29x. When the I/O times were included, GPU-SFFT was still faster than both MIT-SFFTC and cuFFT for all the sparsity levels (Figure 2.3 b)). For this case, the speedup obtained with GPU-SFFT vs MIT-SFFTC had a maximum of 22x and an average of 13x, and the speedup of GPU-SFFT vs cuFFT had a maximum of 5x and an average of 3x.

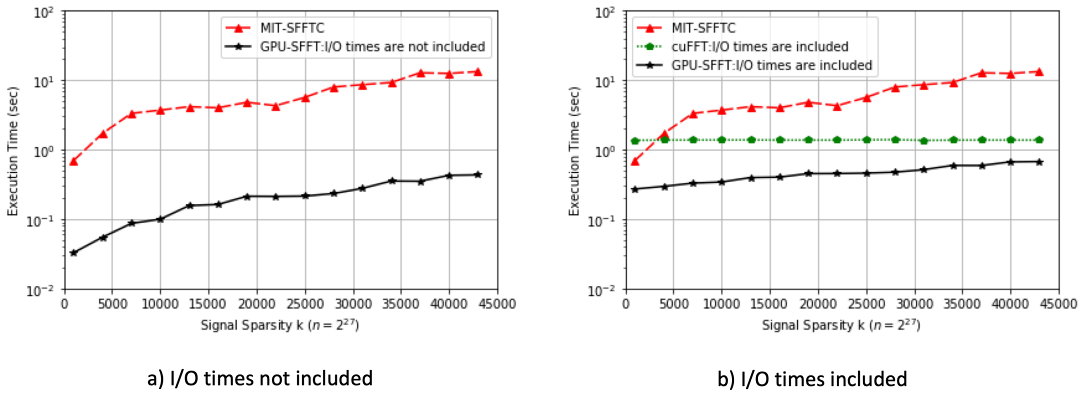


Figure 2.3: GPU-SFFT: Execution Times vs Signal Sparsity

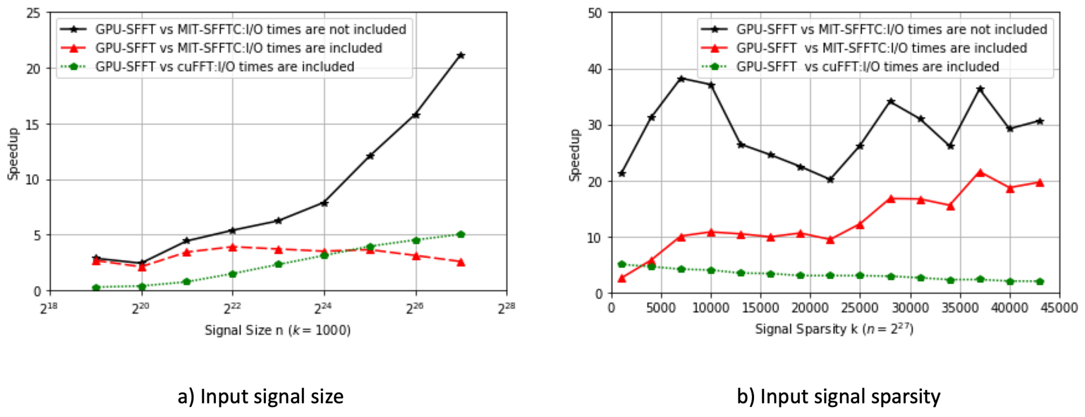


Figure 2.4: GPU-SFFT: Speedup

GPU − SFFT accuracy: Figure 2.5 showed that the mean square error (MSE, Equation 2.4) of the output signal, for both sets of experiments, was below 10^{-4}, with all the k frequencies components in the output of GPU-SFFT successfully recovered. Hence, GPU-SFFT showed a better performance than both the MIT-SFFTC and cuFFT with high levels of accuracy.

2.5 Summary

The Sparse Fast Fourier Transform (MIT-SFFT) is an algorithm to compute the discrete Fourier transform of a signal with a sublinear time complexity, i.e., algorithms

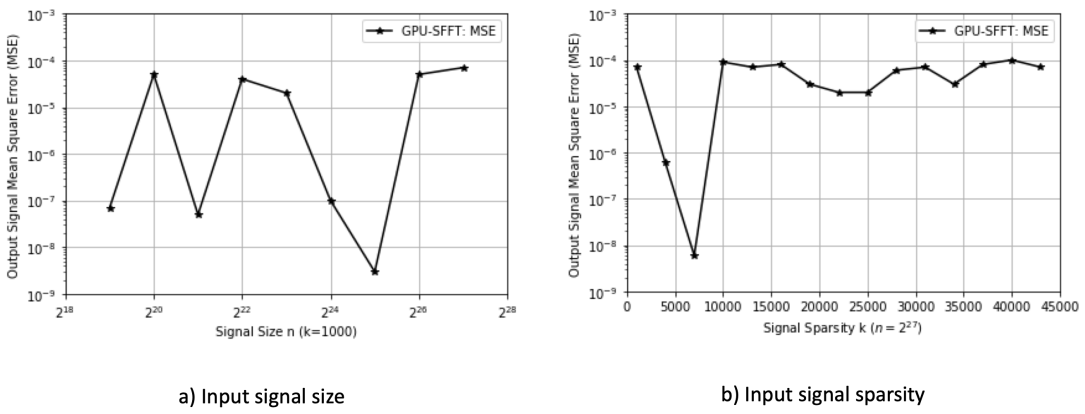


Figure 2.5: GPU-SFFT: Mean Square Errors of the Output Signal

with runtime complexity proportional to the sparsity level k, where k is the number of non-zero coefficients of the signal in the frequency domain. In this chapter, we proposed a highly scalable and parallel CPU-GPU algorithm called GPU-SFFT for computing the SFFT of k-sparse signals. GPU-SFFT was designed to achieve a high-performance algorithm by carefully crafting parallel regions in the sequential MIT-SFFT code [Has18], increasing the parallelism by maximizing the number of concurrent threads executing independent instructions. The design of GPU-SFFT also adopted a specific data-management strategy to minimize the data transfer between the CPU and the GPU leading to reduced I/O times, by transferring only input data from the CPU to the GPU, and only output results from the GPU to the CPU, and performing all the computations on the GPU side.

Our experiments showed that the specific optimizations applied to the design of GPU-SFFT resulted in a substantial decrease in the execution times needed for computing the SFFT. The comparison of the execution times of the GPU-SFFT with the execution times of the sequential MIT-SFFT algorithm showed that the speedup obtained with the GPU-SFFT was up to 38x when the times for data transfer between the CPU and the GPU were not considered, and up to 22x when these times were included. Moreover, up to 5x speedup was obtained when the execution times of the GPU-SFFT were compared to the corresponding times of cuFFT, the NVIDIA CUDA Fast Fourier Transform (FFT) library. For all the experiments performed with the GPU-SFFT algorithm, the mean square error (MSE) was below 10^{-4}, and there were no missing frequencies in the computed SFFT output signal. The source code for GPU-SFFT is available at https://github.com/pcdslab/gpu-sfft.

CUDA Computer Model

The GPU-based high-performance algorithms presented in this chapter and in chapters 3 and 4, were implemented using the following CUDA computer model.

The programmable Graphics Processing Units (GPUs) is a highly parallel, multithreaded, many-core processors with very high computational power and memory bandwidth [Gui20]. The GPU is organized into an array of highly threaded streaming multiprocessors (SMs). Each SM contains several cores that share control logic and instruction cache. The GPU architecture is called Single Instruction Multiple Thread (SIMT), on which hundreds of threads on each core can concurrently execute the same instruction [CM14].

In 2006, NVIDIA introduced the Compute Unified Device Architecture (CUDA), a general-purpose parallel computing platform and programming model that allows programmers to use CUDA-enabled GPUs to solve many complex computational problems [Gui20]. A key component of the CUDA programming model is the kernel, the code that implements the instructions to be executed by the threads on the GPU device. The threads executing the instructions in a kernel are grouped into one or two-dimensional *grids.* The grids are made up of threads *blocks.* The threads in a block are organized in groups of 32 called *warps.* All the threads in a warp execute the same instruction at the same time.

The CUDA memory model contains different types of programmable memory spaces: Registers, shared memory, local memory, constant memory, texture memory, and global memory. Each thread has private local memory. Each thread block has shared memory visible to all threads of the block and with the same lifetime as the block. Using shared memory improves the performance when threads inside the block need to access the same data multiple times. All threads have access to the same global memory. Global memory is the largest, highest latency, and most used memory on a GPU. Data transferred from CPU to GPU resides in global memory. Transferring data between CPU and GPU is a slow process with a negative impact on the performance of a CUDA code, hence this type of transfer should be minimized. Coalesced memory accesses occur when all the 32 threads in warp access adjacent memory locations. Ensuring coalesced global memory access is an important goal for high-performance GPU-based algorithms [CM14].

CHAPTER 3

TURBOBFS: A GPU BASED BREADTH-FIRST SEARCH (BFS) ALGORITHMS

In this chapter, we present the design and implementation of TurboBFS, a highly scalable GPU-based set of top-down and bottom-up BFS algorithms in the language of linear algebra [AS21d].

3.1 Introduction

Graphs that are used for modeling of human brain [BS17], omics data [GOB+10], or social networks [JWBW16], are huge, making a manual inspection of these graphs practically impossible. A popular, and fundamental, method used for making sense of these large graphs is the well-known Breadth-First Search (BFS) algorithm with many interesting applications. However, the high complexity of BFS algorithms is a severe bottleneck for numerous computational problems. Due to its importance, and high computational complexity with numerous applications; the BFS algorithm has been used to evaluate the performance of practically all the high-performance graphs processing libraries such as Ligra for shared memory machines [SB13], Gunrock on the GPU [WDP+16], the implementation for the GraphBLAS standard, SuiteSparse:GraphBLAS [ACD+20], and the GPU based GraphBLAST library [YBO22] built over the GrapBLAS library. Furthermore, the Graph 500 benchmark uses BFS as one of the algorithms for ranking supercomputers [BFG+06].

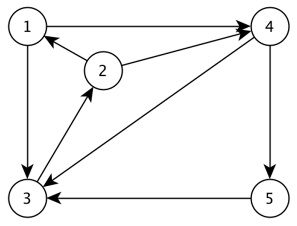
BFS algorithms and their applications are both interesting and challenging for Graphics Processing Units (GPU's) [Gui20], because these algorithms (and architecture) have enough parallelism, but the data-access patterns are highly irregular. Other challenges for implementing BFS in a scalable fashion include limited global memory of the GPU's, the data-transfer PCIe bottleneck, and warp divergence on the GPU kernels. These challenges are primary reasons for limits on the size of the graph that can be processed in a scalable fashion by the available BFS algorithms, and to elicit limited speedups on CPU-GPU architectures, being, therefore, an active area of research [PWW+17, WDP+16, YBO22].

We designed and implemented TurboBFS, a highly scalable set of GPU-based BFS algorithms in the language of linear algebra. As far as we know, the first BFS algorithm in the language of linear algebra was described in chapter 6 of reference [KG11] as a first step of the betweenness centrality algorithm proposed by Brandes [Bra08, KG11]. This algorithm was implemented on the SuiteSparse:GraphBLAS and GraphBLAST libraries. The performances of both GPU-Based BFS algorithms were compared in reference [YBO22].

The algorithms on TurboBFS are applicable to unweighted, directed, and undirected graphs represented by sparse adjacency matrices in the Compressed Sparse Column (CSC) format, or in the transpose of the Coordinate (COO) format, which were equally applicable to direct and undirected graphs. The design of TurboBFS was based on parallel optimizations selected to solve some of the problems associated with the challenges that we just discussed. We implemented a top-down BFS algorithm exploiting the sparsity of the frontier vector, which contains the number of shortest paths from the discovered vertices to the connected undiscovered vertices. Further exploitation of the sparsity is acquired by using the sparsity of the output vector, which contains the number of shortest paths from the root vertex to the discovered vertices. We also implemented the bottom-up BFS algorithm presented in reference [BAP12], as well as an algorithm that combined both approaches [BBAP13], which showed the best performance for some groups of graphs. To optimize the use of the limited global memory of the GPU, we considered all the graphs unweighted, i.e., represented by Boolean sparse adjacency matrices [KG11], so that, the value arrays of the corresponding sparse formats were not stored. This optimization resulted in a substantial reduction of the GPU global memory footprint required by the algorithms, resulting in substantial bandwidth utilization, as well as an increased performance due to a reduction in the number of unnecessary floating operations. Our algorithms were also designed to use only one type of sparse compressed format with the corresponding reduction in the memory footprint. A comprehensive experimental detail and results are presented to assess the performance of the algorithms in TurboBFS.

Regular and irregular graphs

To implement the TurboBFS algorithms, the graphs were classified into two classes: *regular* graphs and *irregular* graphs. The regular graphs were those with a degree distribution with a regular pattern, i.e., with relatively low values of maximum, mean, and variance, while irregular graphs were graphs having some vertices with maximum degrees that are many orders of magnitude greater than their mean, and standard deviations relatively larger than those shown by regular graphs. Figure 3.2 shows the differences between the degree distribution of these two classes of graphs. This Figure shows the relative histogram for the degree distribution of a regular graph, delaunay23, with a maximum, mean, and standard deviation equal to 28, 6.0, and 1.0 respectively, as well as the relative histogram for the degree distribution of an irregular graph, mycielskian19, with a maximum, mean, and standard deviation of their degree distribution equal to 196607, 2297 and 4530 respectively. The dispersion of the degree distribution for the irregular graph is as expected much greater than the corresponding dispersion for the regular graph. In our experiments, we found that the depth d of the BFS tree for irregular graphs is much lower than the depth for regular graphs. A greater depth means that the runtime of the BFS algorithm increases, for example, for the delaunay23 graph, with 8.4×10^6 vertices and 50×10^6 edges, d was equal to 1213 and the runtime 2014 ms, while for the mycielskian19 graph, with 393215 vertices and 903×10^6 edges, d was equal to 3 and the runtime 31 ms. Therefore, the topology of the graphs which determines the depth of the BFS trees had a huge impact on the performance of the TurboBFS algorithms.



$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

CSC

$$\mathbf{row_A} = [\mathbf{2} \quad \mathbf{3} \quad \mathbf{1} \quad \mathbf{4} \quad \mathbf{5} \quad \mathbf{1} \quad \mathbf{2} \quad \mathbf{4}]$$
$$\mathbf{CP_A} = [\mathbf{1} \quad \mathbf{2} \quad \mathbf{3} \quad \mathbf{6} \quad \mathbf{8} \quad \mathbf{9}]$$

COOC

$$\mathbf{row_A} = [\mathbf{2} \quad \mathbf{3} \quad \mathbf{1} \quad \mathbf{4} \quad \mathbf{5} \quad \mathbf{1} \quad \mathbf{2} \quad \mathbf{4}]$$
$$\mathbf{col_A} = [\mathbf{1} \quad \mathbf{2} \quad \mathbf{3} \quad \mathbf{3} \quad \mathbf{3} \quad \mathbf{4} \quad \mathbf{4} \quad \mathbf{5}]$$

Figure 3.1: TurboBFS: COOC and CSC Sparse Storage Formats

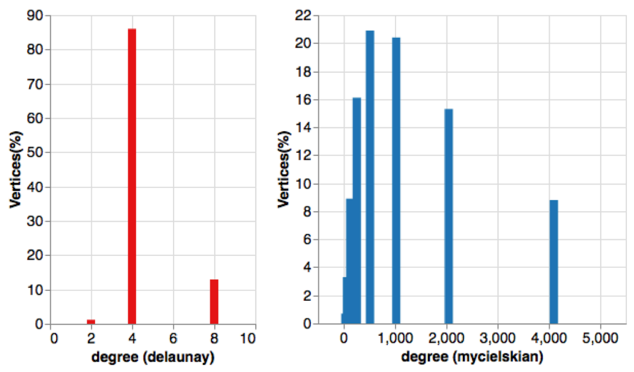


Figure 3.2: TurboBFS: Degree Distribution for Undirected Graphs

3.2 Breadth-First Search (BFS) algorithm

The breadth-first search (BFS) algorithm is applicable to any unweighted, directed or undirected graph $G = (V, E)$, where V is the finite set of vertices and E the set of edges. Any pair $(u, v) \in E$ implies that the vertices u and v in V are connected by an edge in G. A graph G is directed if E consists of ordered pairs, otherwise, G is undirected. Given a graph $G = (V, E)$ and a root vertex $r \in V$, the top-down BFS algorithm performs a systematic search of every vertex on E that is reachable from r. The algorithm computes the shortest path (smallest number of edges) from r to each reachable vertex, producing a BFS tree of the graph G. For connected graphs, the BFS tree is a spanning tree. On every step of the BFS algorithm, the frontier between discovered and undiscovered vertices is expanded. The algorithm discovers all the vertices at depth d, before discovering all the vertices at depth $d + 1$ on the BFS tree. On every step of the top-down BFS there are three sets of vertices. The first set, σ, contains the number of shortest paths from the root vertex to the discovered vertices, the second set, f, contains the number of shortest paths from the discovered vertices to the connected undiscovered vertices, and the third set is the undiscovered vertices, i.e., $V - \sigma - f$. The set f represents the frontier between the discovered and the undiscovered vertices. For a graph G with n vertices and m edges represented by a sparse adjacency matrix, the time complexity of the sequential BFS algorithm is $O(n^2)$ [CLRS22].

In every step of the top-down BFS algorithm, the vertices in f are like parents discovering their children among their neighbors. The step only finishes when each parent has searched for all the potential children. In the bottom-up BFS algorithm [BAP12], the searching process is reversed,i.e., at every step, the undiscovered vertices are like children searching for parents, when a vertex finds a parent among its neighbors, the searching process finishes for that vertex. More details about our implementation of this algorithm are given in Section 3.3.

3.3 TurboBFS algorithms for unweighted graphs

This section presents our versions of the top-down and bottom-up BFS algorithms in the language of linear algebra for directed and undirected unweighted graphs. For the design and implementation of our algorithms, we used the Coordinate column COOC (transpose of the COO format) and the Compressed Sparse Column CSC sparse storage formats to represent the sparse adjacency matrices of the graphs. These formats are the best choice to compute the transpose sparse matrix-vector multiplication ($y = A^T x$) performed in the top-down Algorithm 3.1, as well as the operations needed for the bottom-up Algorithm 3.6. Figure 3.1 shows examples of these formats for a sparse adjacency matrix representing a directed, unweighted graph. For a $n \times n$ adjacency sparse matrix A with nnz non-zero elements representing unweighted graphs, the array $\mathbf{row_A}$ (size nnz) stores the corresponding row indices of these non-zero values. The array $\mathbf{CP_A}$ (size $n+1$) stores the sequential indices (from 1 to nnz) of the non-zero values in A that start a column, the first element of this array is always equal to 1 and the last element equal to $nnz+1$. The array row_A of the COOC format is identical to the corresponding arrays of the CSC format, while the array $\mathbf{col_A}$ (size nnz) stores the corresponding column indices of the non-zero values in A. To reduce the memory footprint and increase the performance of the TurboBFS algorithms, the arrays that store the non-zero values of the sparse adjacency matrix of unweighted graphs were not used by our algorithms.

3.3.1 Top-Down BFS algorithm for COOC or CSC formats

Algorithm 3.1 represents the linear algebra formulation of the top-down BFS algorithm for a graph $G = (V, E)$ with n vertices and nnz edges, represented by $n \times n$

sparse adjacency matrix A in the COOC sparse storage format, with nnz non-zero elements. Algorithm 3.2 is the implementation of the same algorithm for the CSC sparse storage format. These algorithms are inspired by the BFS algorithm described in chapter 6 of reference [KG11], where it is presented as a first step of the linear algebra version of the betweenness centrality algorithm [Bra08]. The main innovation of our algorithms 3.1 and 3.2 was the utilization of the sparsity of the *frontier* vector f and *output* vector σ to substantially improve the performance of the algorithms described in [KG11]. Algorithm 3.1 computes a top-down BFS from

Algorithm 3.1 Linear algebra formulation of the top-down BFS algorithm for an unweighted graph represented by a sparse adjacency matrix A in the COOC format.

```
 1: Input : $\boldsymbol{A}, r, n$.
 2: Output: $\boldsymbol{\sigma}(1....n)$
 3: procedure tdBFS-LA-UG($\boldsymbol{A}$,$\boldsymbol{\sigma}$,$r$,$n$)
 4:     $d \leftarrow 0$
 5:     $c \leftarrow 1$
 6:     $\boldsymbol{f} \leftarrow 0$
 7:     $\boldsymbol{\sigma} \leftarrow 0$
 8:     while $c > 0$ do
 9:         $d \leftarrow d + 1$
10:         $c \leftarrow 0$
11:         if $d == 1$ then
12:             $\boldsymbol{f}(r) \leftarrow 1$
13:             $\boldsymbol{\sigma}(r) \leftarrow 1$
14:         end if
15:         $\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$
16:         $\boldsymbol{f} \leftarrow 0$
17:         if $\exists \boldsymbol{\sigma}(i) == 0$ then
18:             $\boldsymbol{f}(i) \leftarrow \boldsymbol{f_t}(i)$
19:         end if
20:         if $\exists \boldsymbol{f}(i)! = 0$ then
21:             $\boldsymbol{\sigma}(i) \leftarrow \boldsymbol{\sigma}(i) + \boldsymbol{f}(i)$
22:             $c \leftarrow 1$
23:         end if
24:     end while
25: end procedure
```

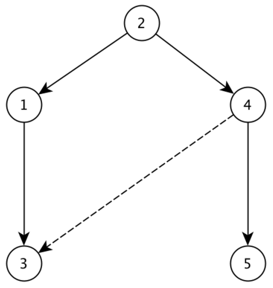
the root vertex r, with d representing the current depth of the discovered vertices. The final value of d is equal to the height of the BFS tree rooted on r. The output vector σ contains the number of shortest paths from the root vertex to the discovered vertices. The frontier vector f contains the number of shortest paths from the discovered vertices to the undiscovered vertices to which there is some edge. The while loop stops when the vector f is equal to 0, i.e., when all the vertices reachable from r have been discovered (lines 20 to 22). The vector f is updated by the sparse matrix-vector multiplication (SpMV) operation with the transpose of the adjacency matrix (line 15), followed by a mask operation (lines 17 to 19) that updates on f the shortest paths to vertices no yet contained on the vector σ, guaranteeing that only the new discovered shortest paths are added to σ (lines 20 and 21).

Algorithm 3.2 represents the linear algebra formulation of the BFS algorithm for a graphs $G = (V, E)$ with the sparse adjacency matrix A in the CSC sparse storage format. This algorithm is similar to Algorithm 3.1, with the difference that due to the properties of the CSC format, the performance of the algorithm is improved by including the mask operation in the SpMV operation, as shown in Algorithm 3.4.

3.3.2 Sparse matrix-vector multiplication

Our experimental results showed that the runtime of the SpMV operation on Algorithms 3.1 and 3.2 can be up to 90 % of their total runtime. Hence the overall performance of these algorithms was mainly determined by the performance of this operation. Algorithm 3.3 implements the sequential SpMV ($\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$) operation of Algorithm 3.1. The parallel version of this algorithm, designated as scCOOC, assigns one thread per edge of the graph. The top-down BFS algorithm using the scCOOC algorithm for the SpMV operation was designated as TurboBFS-tdscCOOC.

Algorithm 3.4 implements the sequential version of the SpMV operation on Algorithm 3.2. The mask operation (line 2) is implemented by computing the components of the vector $\boldsymbol{f_t}$ only when the corresponding component of the σ vector is equal to 0, ensuring that only the new discovered shortest paths are added to σ (lines 19 to 20 of Algorithm 3.2). The straightforward parallelization of Algorithm 3.4, known as CSC-scalar (scCSC), on a GPU kernel assigns one thread per vertex. In this paper, the acronym TurboBFS-tdscCSC designated the top-down BFS algorithm using the scCSC algorithm for the SpMV operation. Figure 3.3 shows the



f	0	1	0	0	0	
σ	0	1	0	0	0	
thread	**1**	**2**	**3**	**4**	**5**	**d**
$f_t \leftarrow fA$	1	0	0	1	0	**1**
$f \leftarrow f_t$	1	0	0	1	0	**1**
$\sigma \leftarrow \sigma + f$	1	1	0	1	0	**1**
$f_t \leftarrow fA$	0	0	2	0	1	**2**
$f \leftarrow f_t$	0	0	2	0	0	**2**
$\sigma \leftarrow \sigma + f$	1	1	2	1	1	**2**
$f_t \leftarrow fA$	0	0	0	0	0	**3**
$f \leftarrow f_t$	0	0	0	0	0	**3**

Figure 3.3: TurboBFS: Example of Computation using the TurboBFS-tdscCSC Algorithm

computation of the top-down BFS for the root vertex 2 of the graph in the example of Figure 3.1, using the TurboBFS-tdscCSC algorithm. The example assigns one thread per vertex and shows the values assigned to the vectors σ and f on each one of the two steps of the BFS computation. For irregular graphs, the scCSC kernel results in poor performance due to uncoalesced memory access and warp divergence. To improve the performance of the SpMV operation for irregular graphs, we implemented the CSC-vector(veCSC) algorithm shown in Algorithm 3.5, which is similar to the CSR-vector algorithm proposed in [BG08]. The veCSC algorithm

Algorithm 3.2 Linear algebra formulation of the top-down BFS algorithm for an unweighted graph represented by sparse adjacency matrices A in the CSC format.

```
1: Input: $\boldsymbol{A}, r, n$.
2: Output: $\boldsymbol{\sigma}(1....n)$
3: procedure BFS-LA-UG($\boldsymbol{A}$,$\boldsymbol{\sigma}$,$r$,$n$)
4:     $d \leftarrow 0$
5:     $c \leftarrow 1$
6:     $\boldsymbol{f} \leftarrow 0$
7:     $\boldsymbol{\sigma} \leftarrow 0$
8:     while $c > 0$ do
9:         $d \leftarrow d + 1$
10:        $c \leftarrow 0$
11:        $\boldsymbol{f_t} \leftarrow 0$
12:        if $d == 1$ then
13:            $\boldsymbol{f}(r) \leftarrow 1$
14:            $\boldsymbol{\sigma}(r) \leftarrow 1$
15:        end if
16:        $\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$
17:        $\boldsymbol{f} \leftarrow \boldsymbol{f_t}$
18:        if $\exists \boldsymbol{f}(i)! = 0$ then
19:            $\boldsymbol{\sigma}(i) \leftarrow \boldsymbol{\sigma}(i) + \boldsymbol{f}(i)$
20:            $c \leftarrow 1$
21:        end if
22:    end while
23: end procedure
```

assigns a warp per vertex. This algorithm incorporates the warp shuffle instruction (lines 18-22) to reduce the local sums by the threads in the warp without using shared memory. The first thread in the warp outputs the result (lines 23-25). The veCSC algorithm solves the problems of no coalesced memory access and the warp divergence of the scCSC algorithm. The best performance of the veCSC algorithm is obtained for irregular graphs, on which the warp divergence is minimized. The acronym TurboBFS-tdveCSC designated the top-down BFS algorithm using the veCSC algorithm for the SpMV operation.

Algorithm 3.3 Algorithm to implement the sequential $\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$ operation of the BFS algorithm 3.1 with the sparse adjacency matrices in the COOC format.

```
1: for $k \to 1, nnz$ do
2:     if $\boldsymbol{f}(\boldsymbol{row_A}(k)) > 0$ then
3:         $\boldsymbol{f_t}(\boldsymbol{col_A}(k)) \leftarrow \boldsymbol{f_t}(\boldsymbol{col_A}(k)) + \boldsymbol{f}(\boldsymbol{row_A}(k))$
4:     end if
5: end for
```

Algorithm 3.4 Algorithm (scCSC) to implement the sequential $\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$ operation of the BFS algorithm 3.2 for sparse adjacency matrix in the CSC format.

```
1: for $i \to 1, n$ do
2:     if $\sigma(i) == 0$ then
3:         $sum \leftarrow 0$
4:         $start \leftarrow \mathbf{CP_A}(i)$
5:         $end \leftarrow \mathbf{CP_A}(i+1) - 1$
6:         for $k \to start, end$ do
7:             $sum \leftarrow sum + \mathbf{f}(\mathbf{row_A}(k))$
8:         end for
9:         if $sum > 0$ then
10:            $\mathbf{f_t}(i) \leftarrow sum$
11:        end if
12:    end if
13: end for
```

3.3.3 Bottom-Up BFS algorithm for the CSC format

Algorithm 3.6 is the linear algebra formulation of the bottom-up BFS algorithm described in reference [BAP12], for an unweighted graph G = (V, E) represented by sparse adjacency matrices A in the CSC sparse storage format. This algorithm computes a bottom-up BFS from the root vertex r, with d representing the current depth of the discovered vertices. The final value of d is equal to the height of the BFS tree rooted on r. The output vector $\boldsymbol{S}$ contains the level d at which each vertex is discovered. If the vertex v is undiscovered (line 12), the search for the parent of v starts at line 15 by searching all the incidents vertices to v (in the vector $\mathbf{I}$), when a parent is found (line 16), the corresponding element of the matrix $\mathbf{S}(v)$ is updated with the next to the current level $(d+1)$. After discovering the parent

Algorithm 3.5 GPU-based algorithm (veCSC) to implement the sparse matrix-vector multiplication $\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$ with the sparse adjacency matrix in the CSC format.

```
 1: Input: $\mathbf{CP_A}$,$\mathbf{row_A}$,$\mathbf{f}$
 2: Output: $\mathbf{f_t}$
 3: procedure veCSC-SpMV-Kernel( $\mathbf{CP_A}$,$\mathbf{row_A}$,$\mathbf{f}$)
 4:     $thread_{id} \leftarrow threadIdx.x + blockIdx.x * blockDim.x$
 5:     $threadLane_{id} \leftarrow thread_{id} * (threadsPerWarp - 1)$
 6:     $warp_{id} \leftarrow thread_{id}/threadsPerWarp$
 7:     $col \leftarrow warp_{id}$
 8:     while $col < n$ do
 9:         if $\sigma(col) == 0$ then
10:             $start \leftarrow \mathbf{CP_A}(col)$
11:             $end \leftarrow \mathbf{CP_A}(col + threadLane_{id})$
12:             $sum \leftarrow 0$
13:             $icp \leftarrow start + threadLane_{id}$
14:             while $icp < end$ do
15:                 $sum \leftarrow sum + \mathbf{f}(\mathbf{row_A}(icp))$
16:                 $icp \leftarrow icp + threadsPerWarp$
17:             end while
18:             $offset \leftarrow threadsPerWarp/2$
19:             while $offset > 0$ do
20:                 $sum \leftarrow sum + \mathrm{shfl} - \mathrm{down} - \mathrm{sync}(offset)$
21:                 $offset \leftarrow offset/2$
22:             end while
23:             if $threadLane_{id} == 0$ then
24:                 $f_t(warp_{id}) \leftarrow sum$
25:             end if
26:         end if
27:         $col \leftarrow col + numWarp$
28:     end while
29: end procedure
```

of v, the searching process is completed with the break instruction on line 17. This early termination of the searching process is the main difference with the top-down BFS algorithms 3.1 and 3.2 on which all the potential children must be checked on each searching step of the algorithm. The straightforward parallelization of Algorithm 3.6 by a GPU kernel assigns one thread per vertex, we used the acronym TurboBFS-busc to designate this algorithm. For irregular graphs, the TurboBFS-busc kernel results in poor performance due to uncoalesced memory access and warp divergence. To improve the performance of Algorithm 3.6 for irregular graphs,

Algorithm 3.6 Linear algebra formulation of the bottom-up BFS algorithm for an unweighted graph represented by sparse adjacency matrices A in the CSC format.

```
1: Input: $\boldsymbol{A}, \boldsymbol{CP_A}, r, n$.
2: Output: $\boldsymbol{S}(1....n)$
3: procedure buBFS-LA-UG($\boldsymbol{A}$,$\boldsymbol{CP_A}$,$r$,$n$)
4:     $d \leftarrow 0$
5:     $c \leftarrow 1$
6:     $\boldsymbol{S} \leftarrow -1$
7:     $\boldsymbol{S}(r) \leftarrow d$
8:     while $c > 0$ do
9:         $c \leftarrow 0$
10:        for $v \rightarrow 1, n$ do
11:            if $\boldsymbol{S}(v) == -1$ then
12:                $k \leftarrow \boldsymbol{CP_A}(v)$
13:                $end \leftarrow \boldsymbol{CP_A}(v+1) - 1$
14:                while $k > end$ do
15:                    if $\boldsymbol{S}(I(k)) == d$ then
16:                        $\boldsymbol{S}(v) \leftarrow d + 1$
17:                        $c \leftarrow 1$
18:                        $break$
19:                    end if
20:                    $k \leftarrow k + 1$
21:                end while
22:            end if
23:        end for
24:        $d \leftarrow d + 1$
25:    end while
26: end procedure
```

we implemented the TurboBFS-buve Algorithm 3.7, which is a simplified version of Algorithm 3.5 and where a warp is assigned to each vertex. The TurboBFS-buve algorithm solves the problems of no coalesced memory access and the warp divergence of the TurboBFS-busc algorithm for irregular graphs. The bottom-up BFS algorithm has the best performance when a large fraction of the vertices are in the frontier [BAP12]. At the beginning of the BFS search, the frontier vector is sparse and the top-down BFS is more efficient than the bottom-up approach. Hence, to yield the best performance, both algorithms can be combined, running the top-down BFS at the beginning of the process, and when the frontier vector

Algorithm 3.7 TurboBFS-buve:GPU-based implementation of Algorithm 3.6, using one warp per vertex.

```
1: Input: $CP_A$,$row_A$,$S$,$d$
2: Output: $S$
3: procedure veCSC-BU-Kernel($CP_A$,$row_A$,$S$,$d$),
4:     $thread_{id} \leftarrow threadIdx.x + blockIdx.x * blockDim.x$
5:     $threadLane_{id} \leftarrow thread_{id} \& (threadsPerWarp - 1)$
6:     $warp_{id} \leftarrow thread_{id}/threadsPerWarp$
7:     $col \leftarrow warp_{id}$
8:     $break \leftarrow 0$
9:     while $!break$ and $col < n$ and $S(col) == -1$ do
10:        $start \leftarrow \mathbf{CP_A}(col)$
11:        $\mathbf{end} \leftarrow \mathbf{CP_A}(\mathbf{col} + \mathbf{threadLane_{id}})$
12:        $icp \leftarrow start + threadLane_{id}$
13:        while $!break$ and $icp < end$ do
14:            if $\mathbf{S}(\mathbf{row_A}(icp)) == d$ then
15:                $\mathbf{S}(col) \leftarrow d$
16:                $c \leftarrow 1$
17:                $break \leftarrow 1$
18:            end if
19:            $icp \leftarrow icp + threadsPerWarp$
20:        end while
21:        $col \leftarrow col + numWarp$
22:    end while
23: end procedure
```

becomes dense to switch to the bottom-up algorithm, in a direction optimizing BFS algorithm [BBAP13]. We designed and implemented a combined BFS algorithm on which the searching process starts with the top-down Algorithm 3.2, and then when the frontier vector f becomes dense, the searching process continues with the bottom-up Algorithm 3.6. We use the heuristics proposed in reference [BBAP13], to switch from the top-down to the bottom-up algorithm when the frontier vector has a minimum of 10% of nonzero elements. We used the acronym TurboBFS-tdbu to designate this combined algorithm.

3.3.4 CUDA implementation of the TurboBFS algorithms

For the CUDA implementation of the TurboBFS algorithms, we designed and implemented Algorithm 3.1 and Algorithm 3.2 with only two kernels on the GPU. The first kernel initializes the f and σ vectors and executes the SpMV($\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$) operation, and the second kernel computes the additional functions of the algorithms. This implementation increased the performance of the algorithm by reducing the overhead due to the sequential execution of more than two kernels on the GPU. The implementation of the bottom-up BFS Algorithm 3.6 used only one GPU kernel.

3.4 Results

The experiments presented in this section were designed to assess the performance of our TurboBFS algorithms by comparing it to the performance of the benchmark algorithms available in the SuiteSparse:GraphBLAS library [ACD+20], and in the GPU-based GraphBLAST [YBO22], and gunrock [WDP+16] libraries.

Our benchmark of sixty-nine graphs used in the experiments are represented by sparse adjacency matrices selected from the SuiteSparse Matrix Collection (formerly the University of Florida Sparse Matrix Collection) [DH11, KAB+19], some of these graphs are also in the Stanford Large Network Dataset Collection [LK14]. The selected adjacency matrices represent thirty-nine undirected and thirty directed graphs, with up to 1900×10^6 edges and up to 214×10^6 vertices. The parameters for each graph are given in the Tables 3.1, 3.3, and 3.5. The weighted graphs were considered unweighted graphs for all the experiments.

The average running time (milliseconds) for each experiment was obtained by 50 trials per experiment. The MTEPs (millions of transverse edges by second) achieved for each BFS algorithm were computed as the ratio between the number of edges (thousands of edges) and the average running time (milliseconds). We also implemented a sequential BFS algorithm to verify the results obtained from the GPU-based algorithms, only the correct results were accepted. For all the results presented in this section, we chose the TurboBFS algorithm with the best performance.

All the experiments presented in this Section were performed on a Linux server with Ubuntu operating system version 16.04.6, 22 Intel Xeon Gold 6152 processors, a clock speed 2.1 GHz, and 125 GB of RAM. The GPU in this server was an NVIDIA Titan Xp, with 30 SM, 128 cores/SM, maximum clock rate of 1.58 GHz, 12196 MB of global memory, and CUDA version 10.1.243 with CUDA capability of 6.1.

3.4.1 Experimental results for regular graphs

This section summarizes the results of the experiments performed for the computation of BFS on thirty-eight regular graphs, nineteen of them direct (D) graphs and the rest undirected (U) graphs. The number of vertices and edges, as well as the parameters (maximum, mean, standard deviation) of the degree (out-degree for directed graphs) distribution of the graphs, are given for each graph in Table 3.1, and Table 3.2 includes the MTPEs and the speedup of the TurboBFS algorithms over the algorithms implemented on the GraphBLAST (**(GBLAST)x**), gunrock (**(gunrock)x**), and SuiteSparse:GraphBLAS (**(GBLAS)x**) libraries. The symbol *OOM* means that the corresponding benchmark algorithm ran out of memory, and the symbol 2.2x in the column (**(gunrock)x**) means that TurboBFS was 2.2x faster than the BFS algorithms in the gunrock libray.

The TurboBFS algorithms obtained up to 2000 MTEPs and were on average 7.4x, 2.0x, and 11.7x faster than the BFS algorithms available on the GraphBLAST, gun-

Table 3.1: Parameters for the set of directed (D) and undirected (U) regular graphs with experimental results given in Table 3.2.

File	$\mathbf{V}\times10^3$	$\mathbf{E}\times10^3$	**degree**($\mathbf{max}/\mu/\sigma$)	**d**
g7jac100sc(D)	30	385	153/13/22	14
g7jac120sc(D)	36	475	153/13/23	14
g7jac140sc(D)	42	566	153/14/24	15
g7jac160sc(D)	47	657	153/14/24	16
g7jac180sc(D)	53.	747	153/14/24	17
g7jac200sc(D)	59	838	153/14/25	17
cit-HepPh(D)	35	422	411/12/15	33
email-Enron(U)	37	368	1383/10/36	10
delaunayn15(U)	33	197	18/6/2	84
delaunayn16(U)	66	393	17/6/2	110
delaunayn17(U)	131	786	17/6/1	157
delaunayn18(U)	262	1573	21/6/1	197
delaunayn19(U)	524	3146	21/6/1	306
astro-ph(U)	17	243	360/15/21	10
ri2010(U)	25	126	44/5/3	66
me2010(U)	70	336	73/5/3	108
az2010(U)	242	1196	137/5/4	128
nc2010(U)	289	1417	83/5/3	207
fl2010(U)	484	2346	177/5/4	151
ca2010(U)	710	3489	141/5/3	216
enron(D)	69	276	1392/4/28	8
Wordnet3(D)	83	133	64/2/2	20
ASIC-100ks(D)	99	579	206/6/6	33
ASIC-320ks(D)	322	1828	412/6/8	31
ASIC-680ks(D)	683	2329	210/3/4	31
smallworld(U)	100	1000	17/10/1	9
luxemb-osm(U)	115	239	6/2/1	1035
netherl-osm(U)	2217	4883	7/2/1	1796
internet(D)	125	207	138/2/4	21
amazon0302(D)	262	1235	5/5/1	72
amazon0312(D)	401	3200	10/8/3	45
amazon0601(D)	403	3387	1/8/3	36
amazon0505(D)	410	3357	10/8/3	36
amazon-2008(D)	735	5158	10/7/4	32
web-NtDame(D)	326	1497	3455/5/22	52
roadNet-PA(U)	1091	3084	9/3/1	542
roadNet-TX(U)	1393	3843	12/31	723
roadNet-CA(U)	1971	5533	12/3/1	555

rock, and SuiteSparse:GraphBLAS libraries respectively. The top-down TurboBFS-tdscCSC algorithm obtained the best performance for twenty-one (55 %) of the graphs, and the bottom-up TurboBFS-busc algorithm showed the best performance for fifteen (40 %) of the graphs in this group.

Table 3.2: Experimental MTEPs and speedup over the GraphBLAST ((**GBLAST)x**), gunrock ((**gunrock)x**), and SuiteSparse:GraphBLAS ((**GBLAS)x**) libraries, obtained with the TurboBFS algorithms for the computation of BFS for the set of regular graphs given in Table 3.1.

File	**MTEPs**	**(GBLAST)x**	**(gunrock)x**	**(GBLAS)x**
g7jac100sc(D)	769	8.4x	2.2x	10.4x
g7jac120sc(D)	792	7.0x	2.0x	8.3x
g7jac140sc(D)	1132	9.0x	2.4x	15.6x
g7jac160sc(D)	1094	8.0x	2.5x	13.3x
g7jac180sc(D)	1068	7.4x	2.3x	13.4x
g7jac200sc(D)	1047	6.2x	2.4x	10.6x
cit-HepPh(D)	703	22.7x	4.2x	13.3x
email-Enron(U)	613	4.5x	1.7x	6.3x
delaunayn15(U)	109	9.0x	3.1x	8.5x
delaunayn16(U)	164	8.9x	3.4x	11.7x
delaunayn17(U)	161	7.6x	2.2x	15.9x
delaunayn18(U)	225	9.1x	2.0x	12.7x
delaunayn19(U)	115	3.6x	1.0x	9.2x
astro-ph(U)	808	15.0x	2.7x	12.7x
ri2010(U)	90	11.1x	3.2x	7.4x
me2010(U)	112	9.4x	2.5x	8.3x
az2010(U)	150	4.1x	1.3x	9.4x
nc2010(U)	104	4.6x	1.2x	9.1x
fl2010(U)	165	3.8x	0.9x	10.6x
ca2010(U)	126	2.9x	0.8x	8.0x
enron(D)	690	8.8x	2.0x	13.0x
Wordnet3(D)	190	10.6x	2.3x	6.6x
ASIC-100ks(D)	482	9.0x	2.7x	14.2x
ASIC-320ks(D)	870	7.6x	1.8x	17.1x
ASIC-680ks(D)	776	5.1x	1.4x	21.7x
smallworld(U)	2000	8.0x	2.4x	24.2x
luxemb-osm(U)	20	16.8x	6.7x	3.5x
netherl-osm(U)	26	2.5x	0.8x	1.9x
internet(D)	345	8.3x	3.2x	29.2x
amazon0302(D)	363	6.8x	2.1x	11.8x
amazon0312(D)	593	3.5x	0.9x	9.1x
amazon0601(D)	847	3.9x	1.0x	12.0x
amazon0505(D)	839	3.9x	1.0x	13.3x
amazon-2008(D)	992	2.9x	0.8x	14.2x
web-NtDame(D)	454	9.5x	1.6x	8.5x
roadNet-PA(U)	71	4.3x	1.0x	9.5x
roadNet-TX(U)	62	4.0x	0.9x	8.8x
roadNet-CA(U)	87	3.6x	1.0x	10.1x

3.4.2 Experimental results for irregular graphs

This section presents the results of the experiments performed for the computation of BFS on twenty-three irregular graphs, six of them direct graphs, and seven-

Table 3.3: Parameters for the set of directed (D) and undirected (U) irregular graphs with experimental results given in Table 3.4.

File	**V**$\times 10^3$	**E**$\times 10^3$	**degree**(**max**/μ/σ)	**d**
mycielski12(U)	3	407	1535/133/149	3
mycielski13(U)	6	1228	3071/200/247.0	3
mycielski14(U)	12	3696	6143/301/407	3
mycielski15(U)	24	11111	12287/452/664	3
mycielski16(U)	49	33383	24575/679/1080	3
mycielski17(U)	98	100246	49151/1020/1747	3
mycielski18(U)	196.6	300934	98303/1531/2817	3
mycielski19(U)	393	903195	196607/2297/4530	3
EAT-SR(D)	23.2	326	78/14/20	7
kron-logn16(U)	66	4913	17999/75/313	6
kron-logn17 (U)	131.1	10229	29937/78/378	6
kron-logn18(U)	262	21166	49164/81/454	6
kron-logn19(U)	524	43563	80676/83/541	6
kron-logn20(U)	1049	89241	131505/85/641	6
kron-logn21(U)	2097	182084	213906/87/756	6
soc-Epinio1(D)	76	509	1801/7/26	11
Linux-call(D)	324	1209	712/4/6	45
web-Stanf (D)	282	2313	255/8/11	147
com-LiveJ (D)	3998	69362	14815/17/43	14
com-Orkut(U)	3072	234370	33133/76/155	8
soc-LiveJour1(D)	4848	68994	20293/14/36	15
mawi-12345(U)	18571	38040	16.4×10^6/2/3806	11
mawi-20000(U)	35991	74485	32.5×10^6/2/5414	11

teen undirected graphs. The number of vertices and edges, as well as the parameters (maximum, mean, standard deviation) of the degree (out-degree for directed graphs) distribution of the graphs, are given for each graph in Table 3.3, and Table 3.4 includes the MTPEs and the speedup of the TurboBFS algorithms over the algorithms implemented on the GraphBLAST (**(GBLAST)x**), gunrock (**(gunrock)x**), and SuiteSparse:GraphBLAS (**(GBLAS)x**) libraries.

The TurboBFS algorithms obtained up to 40 GTEPs, and were on average 3.0x, 1.3x, and 22.4x faster than the BFS algorithms available on the GraphBLAST, gunrock, and SuiteSparse:GraphBLAS libraries respectively. The top-down TurboBFS-tdveCSC algorithm obtained the best performance for seven (30 %), the bottom-up TurboBFS-buve algorithm for five (21.7 %), and the combined top-down bottom-up TurboBFS-tdbu algorithm for six (26.1 %) of the irregular graphs in Table 3.4. The

Table 3.4: Experimental MTEPs and speedup over the GraphBLAST (**(GBLAST)x**), gunrock (**(gunrock)x**), and SuiteSparse:GraphBLAS (**(GBLAS)x**) libraries, obtained with the TurboBFS algorithms for the computation of BFS for the set of irregular graphs given in Table 3.3.

File	**MTEPs**	**(GBLAST)x**	**(gunrock)x**	**(GBLAS)x**
mycielski12(U)	2036	5.5x	2.2x	7.5x
mycielski13(U)	4092	3.9x	1.8x	12.5x
mycielski14(U)	6159	2.0x	1.5x	23.3x
mycielski15(U)	11111	1.3x	1.7x	30.9x
mycielski16(U)	16691	1.1x	1.5x	36.2x
mycielski17(U)	22277	0.8x	1.5x	42.2x
mycielski18(U)	31347	0.8x	1.7x	48.0x
mycielski19(U)	39778	*OOM*	1.8x	57.3x
EAT-SR(D)	1085	6.3x	2.7x	16.7x
kron-logn16(U)	4466	2.1x	1.2x	28.3x
kron-logn17 (U)	4447	1.7x	0.8x	16.7x
kron-logn18(U)	5292	1.3x	0.9x	30.0x
kron-logn19(U)	5808	1.1x	0.9x	17.3x
kron-logn20(U)	3984	1.1x	1.1x	24.6x
kron-logn21(U)	2642	0.9x	1.2x	15.4x
soc-Epinio1(D)	848	8.0x	2.3x	16.7x
Linux-call(D)	432	6.5x	1.2x	3.2x
web-Stanf (D)	160	6.8x	0.8x	3.3x
com-LiveJ (D)	1286	0.9x	0.8x	9.6x
com-Orkut(U)	1698	0.9x	0.8x	6.4x
soc-LiveJour1(D)	940	1.3x	0.9x	7.5x
mawi-12345(U)	1330	5.7x	0.7x	28.4x
mawi-20000(U)	1357	5.2x	0.7x	29.1x

top-down TurboBFS-tdscCOOC algorithm obtained the best performance for five (21.7 %) of these irregular graphs, including the most irregular graphs, the mawi graphs, in the group, showing that the COOC format is the most suitable format for these highly irregular graphs.

3.4.3 Experimental results for big graphs

This section presents the results of the experiments performed for the computation of BFS on eight relatively big graphs, five of them direct graphs, and three undirected graphs. The number of vertices and edges, as well as the parameters (maximum, mean, standard deviation) of the degree (out-degree for directed graphs) distribution of the graphs, are given for each graph in Table 3.5, and Table 3.6 includes the MTPEs and the speedup of the TurboBFS algorithms over the algorithms implemented on the GraphBLAST (**(GBLAST)x**), gunrock (**(gunrock)x**), and SuiteSparse:GraphBLAS (**(GBLAS)x**) libraries. The directed graph sk-2005

Table 3.5: Parameters for the set of directed (D) and undirected (U) big graphs with experimental results given in Table 3.6.

File	$\mathbf{V}\times 10^6$	$\mathbf{E}\times 10^6$	**degree(max**/μ/σ)	**d**
kmer-P1a(U)	139	298	40/2/1	487
kmer-A2a(U)	171	361	40/2/1	515
kmer-V1r(U)	214	465	8/2/1	342
it-2004(D)	41	1151	9964/28/67	50
twitter7(D)	42	1468	3×10^6/35/2420	13
GAP-twitter(D)	62	1468	3×10^6/24/1990	15
GAP-web(D)	51	1930	12869/38/78	56
sk-2005(D)	51	1949	12870/39/78	54

Table 3.6: Experimental MTEPs and speedup over the SuiteSparse:GraphBLAS (**(GBLAS)x**) library, obtained with the TurboBFS algorithms for the computation of BFS for the set of big graphs given in Table 3.5.

File	**MTEPs**	**(GBLAST)x**	**(gunrock)x**	**(GBLAS)x**
kmer-P1a(U)	110	*OOM*	*OOM*	10.0x
kmer-A2a(U)	109	*OOM*	*OOM*	11.5x
kmer-V1r(U)	240	*OOM*	*OOM*	19.1x
it-2004(D)	892	*OOM*	*OOM*	3.3x
twitter7(D)	1537	*OOM*	*OOM*	10.2x
GAP-twitter(D)	811	*OOM*	*OOM*	8.3x
GAP-web(D)	1038	*OOM*	*OOM*	4.0x
sk-2005(D)	1147	*OOM*	*OOM*	4.6x

in Table 3.5 is the largest graph for which the BFS was computed with our available GPU. The first three graphs of Table 3.5 are regular graphs for which the bottom-up TurboBFS-busc algorithm showed the best performance, and the other five graphs are irregular graphs for which the best performance was obtained with the combined top-down and bottom-up TurboBFS-tdbu algorithm. For all the big graphs on Table 3.6, the BFS algorithms on the gunrock and GraphBLAST libraries ran out of memory (OOM), asserting our optimization strategy of reducing the memory footprint to design and implement our highly scalable TurboBFS algorithms.

Figure 3.4 shows that the greatest speedups of the TurboBC algorithms were ob-

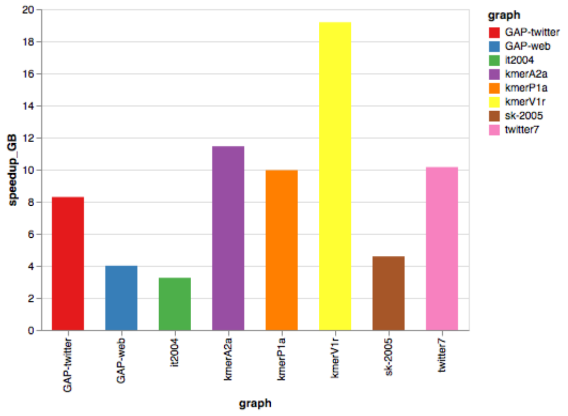


Figure 3.4: TurboBFS: Speedup Experimental Results for Big Graphs

tained for the regular graphs. This Figure also shows that the maximum speedup of the TurboBFS-tdbu algorithm over the BFS algorithm on the GraphBLAS library was obtained for twitter7, the most irregular graph in the group with the smallest value for the depth (d) of the BFS tree. Figure 3.5 shows that the largest value for the MTEPs was obtained for the twitter7 graph, while the smallest values for the METPs were obtained for the regular graphs which had the greatest values for the depth (d) of the BFS tree.

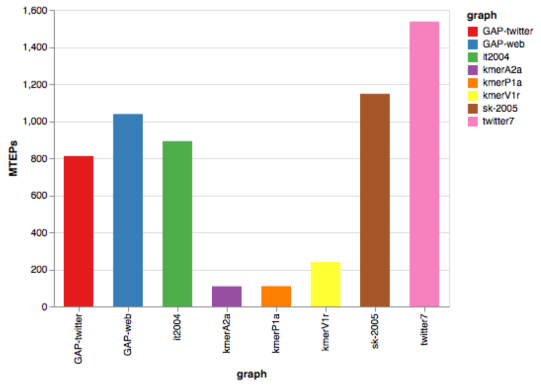


Figure 3.5: TurboBFS: MTEPs Experimental Results for Big Graphs

3.5 Summary

Graphs that are used for modeling the human brain, omics data, or social networks are huge, and manual inspection of these graphs is impossible. A popular, and fundamental, method used for making sense of these large graphs is the well-known Breadth-First Search (BFS) algorithm. However, BFS suffers from large computational cost, especially for big graphs of interest. More recently, the use of Graphics processing units (GPU) has been promising, but challenging because of the limited global memory of GPUs, and irregular structures of real-world graphs.

In this chapter, we presented a GPU-based linear-algebraic formulation and implementation of BFS, called TurboBFS, that exhibits excellent scalability on unweighted, undirected, or directed sparse graphs of arbitrary structure. GraphBLAST and gunrock algorithms use combined top-down bottom-up BFS algorithms that require storing the arrays of the CSC and CSR formats simultaneously on the GPU for directed graphs, increasing the space complexity of the algorithms, and limiting the size of the graphs for which the BFS can be computed. Our approach for designing and implementing the algorithms in TurboBFS differed from the GraphBLAST and the gunrock approaches, because we used highly scalable algorithms which were simpler and hence with less overhead. We also reduced the memory footprint of the TurboBFS algorithms by using only the CSC format for both directed and undirected graphs, and by transferring to the GPU only one set of the arrays that store the indices of the non-zero values of the sparse adjacency matrices representing the graphs, allowing us to compute the BFS for graphs with higher number of edges than those computed by GraphBLAST and the gunrock libraries on the same GPU. This reduction in space complexity also increased the performance of the TurboBFS algorithms.

Our experimental results demonstrated that our TurboBFS algorithms obtained up to 40 GTEPs (billions of transverse edges per second), and were on average 15.7x, 5.8x, and 1.8x faster than the other state-of-the-art BFS algorithms implemented on the sequential SuiteSparse:GraphBLAS, and GPU-based GraphBLAST, and gunrock libraries, respectively. The codes to implement the algorithms proposed in this chapter are available at https://github.com/pcdslab/TurboBFS.

Our future work will be focused on improving the performance of the algorithms in TurboBFS, especially the performance of the algorithms computing the vector sparse matrix multiplication operations. Our goal will be to design and implement GPU-based BFS algorithms with higher performance than the state-of-the-art algorithms available on the GraphBLAST and gunrock libraries.

CHAPTER 4

TURBOBC: A GPU BASED BETWEENNESS CENTRALITY ALGORITHM

In this chapter, we describe the design, implementation, and experimental results computed with TurboBC, a set of GPU-based betweenness centrality (BC) algorithms in the language of linear algebra [AS21c].

4.1 Introduction

Centrality is a fundamental concept in graph analytic [Bav48], used to measure the influence of individual vertices or edges on huge graphs that are used for modeling and analysis of the human brain [RS10], omics data [BPS13], or social networks [OAS10]. One of the most important measures of centrality is the shortest path based *betweenness centrality (BC)*, a metric used to measure the importance of vertices and/or edges in a graph [Fre77].

The BC algorithms have enough parallelism to be implemented on high-performance, parallel graphs processing libraries such as the CPU-based, shared memory, ligra library [SB13]. These algorithms can also be implemented using all the computational power of modern Graphics Processing Units (GPU's) [cor21], however, this implementation is challenging because real-world graphs have some vertices whose degrees are much greater than the mean degree in the graph, resulting in data-access patterns which are highly irregular. This type of data produces load imbalances and warp divergences that negatively affect the performance of the kernels in GPUs. The limited global memory and the data-transfer bottleneck of the GPU are also important challenges to implementing scalable BC algorithms for the BC computation on modern huge graphs. These challenges result in limits in the scalability and performance of the BC algorithms, being therefore an active area of research. One of the first implementations of this algorithm on GPU was given in reference [JLH+12] as an edge-parallel approach, followed by the gpu-fan package described in [SZ11], which was based on an improved All-Pairs Shortest Path (APSP) algorithm. Several hybrid GPU-CPU and multiple GPU implementations are presented in [MB14, SZ11, SKSC13, PWW+17, SB13, WDP+16]. As far as we know, the BC algorithm in the language of linear algebra was first described in chapter 6 of reference [KG11] and implemented in the GraphBLAS library for CPUs [BMM+17], being our proposed TurboBC algorithms, the first GPU-based implementation of BC inspired in this linear algebra algorithm.

The memory-efficient and highly scalable BC algorithms on TurboBC are based on two parallel optimizations. Our first optimization was to reduce the space complexity of the algorithm by limiting the number and the size of the arrays used on the computations performed by the GPU kernels. The second optimization was to design and implement our BC algorithms by exploiting the sparsity of the frontier and output vectors of the Breadth First Search (BFS) stage.

The TurboBC algorithms are applicable to unweighted, directed, and undirected graphs represented by sparse adjacency matrices in the Compressed Sparse Column (CSC) and the transpose of the Coordinate Sparse (COO) formats. The GPU-based gunrock BC algorithms use techniques such as BFS push-pull, that, as illustrated in Figure 4.6, require storing additional auxiliary arrays on the GPU global memory, increasing the space complexity of the algorithms and limiting the size of the graphs for which the BC can be computed with limited memory GPUs. Our approach for designing and implementing the algorithms in TurboBC differed from the gunrock approach because we used memory-efficient and highly scalable algorithms which were simpler and hence with less overhead. To reduce the memory footprint and to increase the memory efficiency and the scalability of TurboBC, the algorithms were designed to use only one sparse storage format for each BC computation, also the number of auxiliary arrays on the device side was minimized. The reduction in the memory footprint increased the memory bandwidth utilization and reduced the number of unnecessary floating operations. The design and implementation of the TurboBC algorithms also exploited the sparsity of the frontier and output vectors of the Breadth First Search (BFS) stage. These optimizations improved the performance and the scalability of the TurboBC algorithms. A comprehensive experimental detail and results are presented to assess the performance of the GPU-based BC algorithms in TurboBC.

4.2 Betweenness centrality algorithm

The shortest-path betweenness centrality (BC) algorithm is applicable to any unweighted, directed or undirected graph $G = (V, E)$, where V is the finite set of vertices and E is the set of edges. Any pair $(u, v) \in E$ implies that the vertices u and v in V are connected by an edge in G. A graph G is directed if E consists of ordered pairs, otherwise, G is undirected. Given a source vertex $s \in V$ in a graph G, the Breadth First Search (BFS) stage of the BC algorithm performs a systematic search of every vertex on E that is reachable from s. The algorithm computes the shortest path, i.e., the smallest number of edges from s to each reachable vertex t. The number of shortest paths between the vertices s and t is denoted by σ_{st}, and $\sigma_{st}(v)$ is equal to the number of shortest paths between s and t passing through the vertex $v \in V$, where v is different than s and t [Bra08, AS21d].

Betweenness centrality of a vertex v, $BC(v)$, in a graph G was formally defined by Freeman [Fre77] as

$$BC(v) = \sum_{s \neq v \neq t} \sigma_{st}(v)/\sigma_{st} = \sum_{s \neq v \neq t} \delta_{st}(v) \tag{4.1}$$

where $\sigma_{st}(v)/\sigma_{st} = 0$, if $\sigma_{st} = 0$, and $\delta_{st}(v) = \sigma_{st}(v)/\sigma_{st}$, the *pair-wise dependences*, is the fraction of shortest paths between the vertices s and t that pass through v. This definition of BC equally applies to disconnected and connected, directed and undirected graphs [Fre77]. The straightforward computation of the BC of a vertex v, starts by computing the number and the length of all pairs' shortest paths over the graph, followed by computing the BC for each vertex by looking at all other pairs of vertices and increasing the value of $BC(v)$ if the vertex, v, was in the corresponding shortest path. If $|V| = n$, the time complexity of this BC algorithm is $O(n^3)$, and its space complexity is $O(n^2)$.

Brandes [Bra08], proposed a more efficient BC algorithm on which the pair-wise dependences can be aggregated without computing all of them explicitly. Let the one-sided dependences be defined as

$$\delta_s(v) = \sum_{t \in V} \delta_{st}(v) \tag{4.2}$$

for all $s, v \in V$. Then

$$BC(v) = \sum_{s \neq v} \delta_s(v) \tag{4.3}$$

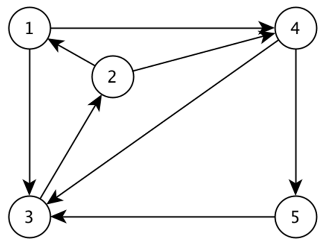
The following recurrence relation computes the one-sided dependences in the Brandes' BC algorithm

$$\delta_s(v) = \sum_{w: d(s,w) = d(s,v)+1} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_s(w)) \tag{4.4}$$

where $d(s, v)$ is the length of the shortest path from s to v. The recurrence relation 4.4 computes the one-sided dependencies of a vertex s on some vertex v from the one-sided dependence of a vertex w one edge far away. For a graph with $|E| = m$, the time complexity of Brandes' algorithm for unweighted graphs is equal to $O(nm)$, and the space complexity is equal to $O(n+m)$. This algorithm is especially suitable for graphs represented by sparse adjacency matrices.

4.3 TurboBC algorithms for unweighted graphs.

This section describes the design and implementation of our GPU-based TurboBC algorithms in the language of linear algebra for unweighted graphs. The TurboBC algorithms were implemented for graphs represented by sparse adjacency matrices in the Compressed Sparse Column (CSC) format, as well as in the COOC format which is the transpose of the Coordinate Sparse (COO) format. Both sparse formats are suitable to implement the sparse matrix-vector multiplication operations included in the BC Algorithm 4.1. Figure 4.1 shows an example of the CSC and COOC formats for a sparse adjacency matrix representing an undirected, unweighted graph. For a $n \times n$ adjacency sparse matrix A with m non-zero elements representing unweighted graphs, the array row_A (size m) of the CSC format, stores the corresponding row indices of the subsequent non-zero values of the columns in the matrix, and the array CP_A (size $n+1$) stores the indices of the elements in the array row_A, that start a column. The first element of CP_A is always equal to 1 (one-based format) and the last element is equal to $m+1$. The COOC format contains two arrays: row_A which is equal to the corresponding array in the CSC format, and the col_A (size m) array that stores the column indices of the non-zero values of the adjacency matrix A. To reduce the memory footprint and increase the performance of the TurboBC algorithms, the arrays that store the non-zero values of the binary sparse adjacency matrix of unweighted graphs were not used in the corresponding sparse matrix-vector multiplication (SpMV) operations of our algorithms.



$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

CSC

$$\mathbf{row_A} = [\mathbf{2} \;\; \mathbf{3} \;\; \mathbf{1} \;\; \mathbf{4} \;\; \mathbf{5} \;\; \mathbf{1} \;\; \mathbf{2} \;\; \mathbf{4}]$$
$$\mathbf{CP_A} = [\mathbf{1} \;\; \mathbf{2} \;\; \mathbf{3} \;\; \mathbf{6} \;\; \mathbf{8} \;\; \mathbf{9}]$$

COOC

$$\mathbf{row_A} = [\mathbf{2} \;\; \mathbf{3} \;\; \mathbf{1} \;\; \mathbf{4} \;\; \mathbf{5} \;\; \mathbf{1} \;\; \mathbf{2} \;\; \mathbf{4}]$$
$$\mathbf{col_A} = [\mathbf{1} \;\; \mathbf{2} \;\; \mathbf{3} \;\; \mathbf{3} \;\; \mathbf{3} \;\; \mathbf{4} \;\; \mathbf{4} \;\; \mathbf{5}]$$

Figure 4.1: Example of CSC and COOC sparse storage formats for a sparse adjacency matrix representing a directed, unweighted graph.

Scalar and vector algorithms for regular and irregular graphs

We implemented two types of BC algorithms. The first type, called scalar algorithms, computes the sparse matrix-vector multiplication with GPU kernels, assigning one thread per vertex (CSC format) or one thread per edge (COOC format). The second type of algorithm, vector algorithms, computes the sparse matrix-vector multiplication with GPU kernels that assign one warp per vertex (CSC format).

In this chapter, the graphs were classified into two classes: *regular* graphs and *irregular* graphs. The regular graphs are those for which, in our experiments, the scalar BC algorithms obtained the best performance, while irregular graphs are those for which the vector BC algorithms obtained the best performance.

We also applied the scale-free metrics, proposed in reference [LAT+05], to approximately quantify when a graph is regular or irregular. The scale-free metrics scf, for a graph $G = (V, E)$ is defined by

$$scf = \sum_{(u,v)\in E} \text{degree(u)} * \text{degree(v)} \tag{4.5}$$

where degree(u) is the degree of vertex $u \in V$, for directed graphs degree(u) = out.degree(u). Our experiments showed that for regular graphs the scf metric is in the range $[1, 224]$, and for irregular graphs in the range $[5846, 651837]$, more details about these results are given in Section 4.4 on which the experimental results are presented.

4.3.1 Brandes' BC algorithm in the language of linear algebra

Algorithm 4.1 represents the linear algebra formulation of the Brandes' BC algorithm for a graph $G = (V, E)$ with n vertices and m edges, represented by $n \times n$ sparse adjacency matrix A in the COOC format, with m non-zero elements. This algorithm is inspired by the BC algorithm described in chapter 6 of reference [KG11]. Algorithm 4.1 computes the exact betweenness centrality vector, $\boldsymbol{bc}$, for all the connected vertices of the graph G using a two-stage procedure.

The first stage of Algorithm 4.1 is a *forward* stage on which a Breadth First Search (BFS) from the source vertex s is performed at the first while loop (lines 11 to 28), where d represents the current depth of the discovered vertices. The final value of d is equal to the height of the BFS tree rooted at s. The output vector $\boldsymbol{\sigma}$ contains the number of shortest paths from the source vertex to the discovered vertices. The frontier vector $\boldsymbol{f}$ contains the number of shortest paths from the discovered vertices in the last iteration, to the undiscovered vertices to which there is some edge. The while loop stops when the vector $\boldsymbol{f}$ is equal to 0, i.e., when all the vertices reachable from s have been discovered. The vector $\boldsymbol{f}$ is updated by the sparse matrix-vector multiplication (SpMV) operation with the adjacency matrix (line 19), followed by a mask operation (lines 20 to 22) that exploits the sparsity of the vector $\boldsymbol{\sigma}$ and updates the shortest paths to vertices on $\boldsymbol{f}$, not yet contained on the vector $\boldsymbol{\sigma}$, guaranteeing that only the new discovered shortest paths are added to $\boldsymbol{\sigma}$ (line 25). By using the sparsity of the vector $\boldsymbol{f}$, the vectors $\boldsymbol{S}$ and $\boldsymbol{\sigma}$ are updated only when the corresponding component of the vector $\boldsymbol{f}$ is not zero (lines 23 to 27). The vector $\boldsymbol{S}$ stores the depth at which each vertex is discovered.

The second stage of Algorithm 4.1 is a *backward* stage on which the one-sided dependences vector, $\boldsymbol{\delta}$, is computed within the second while loop (lines 31 to 42), using Equation 4.4. For the computations in this stage, the vertices are visited in reverse order of their depth. The computation of the vector $\boldsymbol{\delta}$ starts when the auxiliary vector $\boldsymbol{\delta_u}$ is computed (lines 32 to 36) for those values derived from the children at depth d, which are stored on the vector S. The vector $\boldsymbol{\delta_u}$ is then weighted by the adjacency matrix A with the SpMV operation (line 37). The vector $\boldsymbol{\delta}$ is updated (lines 38 to 40), with the values corresponding at depth $d-1$ as determined by the vector $\boldsymbol{S}$. Finally, the betweenness centrality vector $\boldsymbol{bc}$ is computed, using Equation 4.3, for all parent vertices, v, not equal to the source vertex s (lines 43 to 47). For undirected graphs the computation of the vector $\boldsymbol{bc}$ should compensate by the double counting of every pair of vertices, hence $bc(v) \leftarrow bc(v) + \delta(v)/2$ for these graphs [Bra08].

The BC algorithm with the sparse adjacency matrix in the CSC format has the same two stages of Algorithm 4.1, with the difference that in the first stage, the mask operation is included in the SpMV operation as shown in Algorithm 4.3.

Algorithm 4.1 Linear algebra shortest path vertex betweenness centrality algorithm for a graph represented by a sparse adjacency matrix A in the COOC sparse storage format.

```
1: Input: $\boldsymbol{A}$.                          ▷ sparse adjacency matrix representing a graph.
2: Output: $\boldsymbol{\sigma}(1....n)$             ▷ stores number of shortest paths.
3: Output: $\boldsymbol{bc}(1....n)$                 ▷ betweenness centrality vector
4: procedure BC-LA($G = \boldsymbol{A} : \mathbb{B}^{n\times n}$)
5:     $\boldsymbol{bc} \leftarrow 0$
6:     for $s \leftarrow 1, n$ do                    ▷ $s$: source vertex of BFS tree
7:         $d \leftarrow 0$                          ▷ $d$: the current depth being examined
8:         $c \leftarrow 1$                          ▷ c: check if the vector $\boldsymbol{f}$ is equal to 0
9:         $\boldsymbol{S} \leftarrow 0$             ▷ stores depth at which a vertex is discovered
10:        $\boldsymbol{\sigma} \leftarrow 0$
11:        while $c > 0$ do                          ▷ BFS stage starts
12:            $d \leftarrow d + 1$
13:            $c, \boldsymbol{f_t} \leftarrow 0$
14:            if $d == 1$ then
15:                $f(s), \sigma(s) \leftarrow 1$
16:            end if
17:            $\boldsymbol{f_t} \leftarrow \boldsymbol{A^T f}$
18:            if $\exists\sigma(i) == 0$ then
19:                $f(i) \leftarrow f_t(i)$
20:            end if
21:            if $\exists f(i)! = 0$ then
22:                $S(i) \leftarrow d$
23:                $\sigma(i) \leftarrow \sigma(i) + f(i)$
24:                $c \leftarrow 1$
25:            end if
26:        end while
27:        $d \leftarrow d - 1$
28:        $\boldsymbol{\delta} \leftarrow 0$
29:        while $d > 1$ do                          ▷ one-sided dependences vector stage starts
30:            $\boldsymbol{\delta_u}, \boldsymbol{\delta_{ut}} \leftarrow 0$
31:            if $S(i) == d$ and $\sigma(i) > 0$ then
32:                $\delta_u(i) \leftarrow (1.0 + \delta(i)) \div \sigma(i)$
33:            end if
34:            $\boldsymbol{\delta_{ut}} \leftarrow \boldsymbol{A^T \delta_u}$
35:            if $S(i) == d - 1$ then
36:                $\delta(i) \leftarrow \delta(i) + \delta_{ut}(i) \times \sigma(i)$
37:            end if
38:            $d \leftarrow d - 1$
39:        end while
40:        for $v \leftarrow 1, n$ do                ▷ update of vector bc starts
41:            if $v \neq s$ then
42:                $bc(v) \leftarrow bc(v) + \delta(v)$
43:            end if
44:        end for
45:    end for
46:    return $\boldsymbol{bc}$
47: end procedure
```

4.3.2 Sparse matrix-vector multiplication (SpMV).

Our experimental results showed that the runtime of the SpMV operation (lines 19 and 37) can be up to 90 % of the total runtime of Algorithm 4.1, determining, therefore, the overall performance of the BC algorithm. We implemented the SpMV operation with three algorithms, the first one based on the COOC format and the other two based on the CSC format.

Algorithm 4.2 Algorithm to implement the sequential SpMV operations of Algorithm 4.1 (lines 19 and 37) with the sparse adjacency matrix in the COOC format.

```
1: Input: $\boldsymbol{x}$,$\boldsymbol{row_A}$,$\boldsymbol{col_A}$
2: Output: $\boldsymbol{y}$
3: procedure scCOOC-SpMV($\boldsymbol{x}$,$\boldsymbol{row_A}$,$\boldsymbol{col_A}$,$\boldsymbol{y}$)
4:     for $k \to 1, m$ do
5:         if $x(row_A(k)) > 0$ then
6:             $y(col_A(k)) \leftarrow y(col_A(k)) + x(row_A(k))$
7:         end if
8:     end for
9: end procedure
```

There are graphs with some vertices with a much higher degree than the mean value of the degrees in the graph, the SpMV operation for these graphs creates load unbalance in the threads of the GPU which negatively affects the performance of the SpMV algorithm. Our experiments showed that the SpMV algorithm based on the COOC format is less affected by this load unbalance, when applied to regular graphs that have vertices with much higher degrees than the mean degree of the graph. Algorithm 4.2 implements the sequential version of the SpMV operations on the first and second stages (lines 19 and 37) of Algorithm 4.1 with the sparse adjacency matrix in the COOC format. The sparsity of vector $\boldsymbol{x}$ is exploited by updating the vector $\boldsymbol{y}$ only when the corresponding component of vector $\boldsymbol{x}$ is greater than zero (line 5). The parallelization of Algorithm 4.2, known as COOC-scalar (scCOOC), on a GPU kernel assigns one thread per edge. The acronym TurboBC-scCOOC designates the BC algorithm using the scCOOC algorithm for the SpMV operation.

Algorithm 4.3 Algorithm to implement the sequential SpMV operations of Algorithm 4.1 (lines 19 and 37) with the sparse adjacency matrix in the CSC format.

```
1: Input: $\boldsymbol{x}$,$\boldsymbol{CP_A}$,$\boldsymbol{row_A}$
2: Output: $\boldsymbol{y}$
3: procedure scCSC-SpMV($\boldsymbol{x}$,$\boldsymbol{CP_A}$,$\boldsymbol{row_A}$,$\boldsymbol{y}$)
4:     for $i \to 1, n$ do
5:         if $\sigma(i) == 0$ then
6:             $sum \leftarrow 0$
7:             $start \leftarrow CP_A(i)$
8:             $end \leftarrow CP_A(i+1) - 1$
9:             for $k \to start, end$ do
10:                $sum \leftarrow sum + x(row_A(k))$
11:            end for
12:            if $sum > 0$ then
13:                $y(i) \leftarrow sum$
14:            end if
15:        end if
16:    end for
17: end procedure
```

Our experiments showed that for some medium-sized graphs (see Table 4.2), the best performance was obtained with Algorithm 4.3, which implements the sequential version of the SpMV operations on Algorithm 4.1, with the sparse adjacency matrix in the CSC format. Algorithm 4.3 implements the mask operation (line 5) by computing the components of the vector $\boldsymbol{y}$ only when the corresponding component of the $\boldsymbol{\sigma}$ vector is equal to 0, ensuring that only the new discovered shortest paths are added to $\boldsymbol{\sigma}$ (line 25 of Algorithm 4.1). The sparsity of vector $\boldsymbol{x}$ is used on line 12 when the vector $\boldsymbol{y}$ is updated only when the variable *sum* is greater than zero. The straightforward parallelization of Algorithm 4.3, known as CSC-scalar (scCSC), with a GPU kernel, assigns one thread per vertex. The acronym TurboBC-scCSC designates the BC algorithm using the scCSC algorithm for the SpMV operation.

Algorithm 4.4 GPU-based algorithm to implement the SpMV (veCSC) operation of Algorithm 4.1 (lines 19 and 37) with the sparse adjacency matrix in the CSC format.

```
1: Input: $\boldsymbol{x}$,$\boldsymbol{CP_A}$,$\boldsymbol{row_A}$
2: Output: $\boldsymbol{y}$
3: procedure veCSC-SpMV-Kernel($\boldsymbol{x}$,$\boldsymbol{CP_A}$,$\boldsymbol{row_A}$,$\boldsymbol{y}$)
4:     $thread_{id} \leftarrow threadIdx.x + blockIdx.x * blockDim.x$
5:     $threadLane_{id} \leftarrow thread_{id} \& (threadsPerWarp - 1)$
6:     $warp_{id} \leftarrow thread_{id}/threadsPerWarp$
7:     while $col < n$ do
8:         if $\sigma(col) == 0$ then
9:             $start \leftarrow CP_A(warp_{id})$
10:            $end \leftarrow CP_A(warp_{id} + threadLane_{id})$
11:            $sum \leftarrow 0$
12:            $icp \leftarrow start + threadLane_{id}$
13:            while $icp < end$ do
14:                $sum \leftarrow sum + y(row_A(icp))$
15:                $icp \leftarrow icp + threadsPerWarp$
16:            end while
17:            $offset \leftarrow threadsPerWarp/2$
18:            while $offset > 0$ do
19:                $sum \leftarrow sum + \mathrm{shfl} - \mathrm{sync}(mask, sum, offset)$
20:                $offset \leftarrow offset/2$
21:            end while
22:            if $threadLane_{id} == 0 \wedge sum > 0$ then
23:                $y(warp_{id}) \leftarrow sum$
24:            end if
25:        end if
26:        $col \leftarrow col + num - warps$
27:    end while
28: end procedure
```

Our experiments for irregular graphs showed that both the TurboBC-scCOOC and the TurboBC-scCSC algorithms resulted in poor performance due to uncoalesced memory access and warp divergence. To improve the performance of the SpMV operation for irregular graphs, we implemented the CSC-vector(veCSC) algorithm shown in Algorithm 4.4, which is similar to the CSR-vector algorithm proposed in [BG08]. The veCSC algorithm assigns a warp for vertex. This algorithm incorporates the warp shuffle instruction (lines 18-21) to reduce the local sums by the threads in the warp without using shared memory. The first thread in the warp outputs the final result (lines 22-24). The veCSC algorithm solves the problems of no coalesced memory access and warp divergence of the scalar algorithms when applied to irregular graphs. The best performance of the veCSC algorithm is obtained for irregular graphs, on which the warp divergence is minimized. The acronym TurboBC-veCSC designates the BC algorithm using the veCSC algorithm for the SpMV operation.

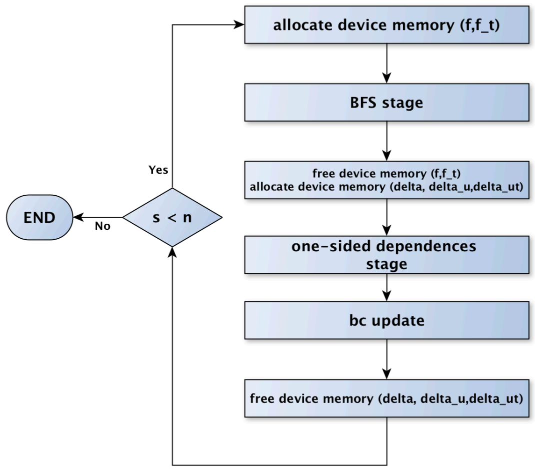

allocate device memory (f,f_t)
BFS stage
free device memory (f,f_t)
allocate device memory (delta, delta_u,delta_ut)
one-sided dependences stage
bc update
free device memory (delta, delta_u,delta_ut)
s < n
Yes
No
END

Figure 4.2: Pipeline for the CUDA implementation of Algorithm 4.1.

4.3.3 CUDA implementation of the TurboBC algorithm

We designed and implemented Algorithm 4.1 using the pipeline shown in Figure 4.2. The BFS stage of Algorithm 4.1 was implemented using two kernels, the first kernel initializes the $\boldsymbol{f}$ and $\boldsymbol{\sigma}$ vectors (lines 15 to 18) and executes the SpMV ($\boldsymbol{f_t} \leftarrow \boldsymbol{f}\boldsymbol{A}$) operation (line 19), the second kernel computes the additional functions of this stage. The computation of the one-sided dependences vector, $\boldsymbol{\delta}$, was implemented using three kernels, the first kernel updates the vector $\boldsymbol{\delta_u}$ (lines 34-36), the second kernel computes the SpMV operation (line 37), and the third kernel updates the vector $\boldsymbol{\delta}$ (lines 38-40). One additional kernel updates the vector $\boldsymbol{bc}$ (lines 43 to

47). This implementation increased the performance of the algorithm by reducing the overhead due to the sequential execution of too many kernels on the GPU. Our experiments showed that the performance of Algorithm 4.1 increased when the SpMV operation of the BFS stage was performed over integer data types for the $\boldsymbol{f}$ and $\boldsymbol{f_t}$ vectors. Hence, in order to minimize the memory footprint on the GPU due to the storage of auxiliary vectors, the deallocation of the device memory for the vectors $\boldsymbol{f}$ and $\boldsymbol{f_t}$ is followed by the allocation of device memory to the float type vectors $\boldsymbol{\delta}, \boldsymbol{\delta_u}$, and $\boldsymbol{\delta_{ut}}$. The SpMV operation with integer data types was up to 2.7x faster than the same operation with float data types, with a very small overhead due to the allocation and deallocation operations of the device memory.

4.4 Results

The experiments presented in this section were designed to assess the performance of our TurboBC algorithms by comparing them to the benchmark parallel BC algorithms available in the state-of-the-art GPU-based gunrock [WDP+16] and CPU-based, shared memory, ligra [SB13] libraries. We also compared the performance of the TurboBC algorithms with the performance of our implementation of the sequential version of Algorithm 4.1 with the sparse adjacency matrix in the CSC format.

Our benchmark of thirty-three graphs used in the experiments were represented by sparse adjacency matrices selected from the SuiteSparse Matrix Collection (formerly the University of Florida Sparse Matrix Collection) [KAB+19], and from the Stanford Large Network Dataset Collection [LS16]. The selected adjacency matrices represented eighteen undirected and fifteen directed graphs, covering a wide range of vertices $[28 \times 10^3, 214 \times 10^6]$ and edges $[171 \times 10^3, 1950 \times 10^6]$. The parameters for the selected graphs are given in Tables 4.1, 4.3, 4.5, and 4.7. The weighted graphs were considered unweighted graphs for all the experiments.

The average runtime (milliseconds) for each experiment was obtained by 50 trials per experiment. We used the sequential version of the BC algorithm to verify the results obtained from the TurboBC algorithms, only the correct results were accepted. For all the results presented in this section, we chose the TurboBC algorithm which showed the best performance for each graph. For the experiments on which the BC was computed for one vertex, the MTEPs (millions of transverse edges by second), achieved for the BC algorithms, were computed as the ratio m/t where m is the number of edges (thousands) and t is the average runtime (milliseconds). For the experiments on which the exact BC was computed for all the vertices in the graph, the MTEPs were computed as mn/t where n is the number of vertices (t in seconds, mn in millions).

All the experiments presented in this section were performed on a Linux server with Ubuntu operating system version 16.04.6, 22 Intel Xeon Gold 6152 processors, a clock speed 2.1 GHz, and 125 GB of RAM. The GPU in this server was an NVIDIA Titan Xp, with 30 SM, 128 cores/SM, maximum clock rate of 1.58 GHz, 12196 MB of global memory, and CUDA version 10.1.243 with CUDA capability of 6.1.

4.4.1 Experimental results for regular graphs

This section summarizes the results of the experiments performed with the TurboBC algorithms to compute the BC of one vertex on twenty regular graphs, twelve of them directed (D) graphs and eight undirected (U) graphs. The number of vertices (n) and edges (m), the parameters (maximum, mean, standard deviation) of the degree (out-degree for directed graphs) distribution, the depth of the BFS tree (d), and the scale free metrics (scf), are given for each graph in Tables 4.1 and 4.3, and Tables 4.2 and 4.4 include the runtime, MTPEs and the speedup obtained by the TurboBC algorithms over the algorithms implemented on the gunrock (**(gunrock)x**) and ligra (**(ligra)x**) libraries, and over the sequential algorithm (**(seq.)x**). The symbol 2.7x in the column (**(gunrock)x**) means that TurboBC was 2.7x faster than the BC algorithms in the gunrock library.

The TurboBC-scCSC algorithm showed the best performance by obtaining the

Table 4.1: Parameters for the set of directed (D) and undirected (U) regular graphs with experimental results given in Table 4.2.

File	**n**$\times10^3$	**m**$\times10^3$	**degree**(**max**$/\mu/\sigma$)	**d**	**scf**
mark3j060sc(D)	28	171	44/6/4	42	10
mark3j080sc(D)	37	228	44/6/4	52	10
mark3j100sc(D)	46	285	44/6/4	62	10
mark3j120sc(D)	55	343	44/6/4	72	10
g7j140sc(D)	42	566	153/14/24	15	197
g7j160sc(D)	47	657	153/14/24	16	208
delaunayn15(U)	33	197	18/6/1	84	13
delaunayn16(U)	66	393	17/6/1	110	14
luxemb-osm(U)	115	239	6/2/0	1035	2
internet(D)	125	207	138/2/4	21	1

Table 4.2: Experimental runtime, MTPEs and the speedup obtained by the TurboBC-scCSC algorithm over the algorithms implemented on the gunrock (**(gunrock)x**) and ligra (**(ligra)x**) libraries, and over the sequential algorithm (**(seq.)x**), with the computation of BC/vertex for the set of regular graphs given in Table 4.1.

File	**runtime**	**MTEPs**	**(seq.)x**	**(gunrock)x**	**(ligra)x**
mark3j060sc(D)	2.1	82	11.5x	2.7x	2.2x
mark3j080sc(D)	2.8	82	9.8x	2.5x	1.5x
mark3j100sc(D)	3.5	82	11.4x	2.4x	1.5x
mark3j120sc(D)	4.4	78	12.9x	2.2x	1.6x
g7j140sc(D)	1.2	472	12.5x	1.9x	2.3x
g7j160sc(D)	1.4	469	13.3x	1.8x	2.6x
delaunayn15(U)	4.7	42	14.4x	2.4x	1.2x
delaunayn16(U)	7.1	55	25.3x	2.2x	1.9x
luxemb-osm(U)	50.0	5	24.7x	2.3x	1.0x
internet(D)	1.5	138	37.8x	1.9x	2.0x

maximum values of speedup and MTEPs, for the ten regular graphs in Table 4.2, obtaining up to 472 MTEPs, as well as a maximum of 37.8x and an average of

17.4x speedup over the sequential code, a maximum of 2.7x and an average of 2.2x speedup over the the BC algorithm available in the gunrock library, and a maximum of 2.6x and an average of 1.8x speedup over the BC algorithm available in the ligra library. The scale-free metrics *scf* for this group of regular graphs varied in the range [1,208] and 80 % of the graphs had a value below 15 for this metric. The depth (d) of the BFS tree was below 100 for 80 % of the graphs in this group.

For the ten regular graphs in Table 4.4, the TurboBC-scCOOC algorithm showed

Table 4.3: Parameters for the set of directed (D) and undirected (U) regular graphs with experimental results given in Table 4.4.

File	**n**$\times 10^3$	**m**$\times 10^3$	**degree(max**/μ/σ)	**d**	**scf**
g7j180sc(D)	53	747	153/14/24	17	217
g7j200sc(D)	59	838	153/14/25	18	224
mark3j140sc(D)	64	400	44/6/4	82	10
smallworld(U)	100	1000	17/10/1	9	61
ASIC-100ks(D)	99	579	206/6/6	33	3
ASIC-680ks(D)	683	2329	210/3/4	31	2
com-Youtube(U)	1135	5975	28754/5/51	14	8
mawi-12345(U)	18571	38040	$16 \times 10^6/2/3806$	10	2
mawi-20000(U)	35991	74485	$33 \times 10^6/2/5414$	11	2
mawi-20030(U)	68863	143415	$63 \times 10^6/2/7597$	12	2

Table 4.4: Experimental runtime, MTPEs and the speedup obtained by the TurboBC-scCOOC algorithm over the algorithms implemented on the gunrock (**(gunrock)x**) and ligra (**(ligra)x**) libraries, and over the sequential algorithm (**(seq.)x**), with the computation of BC/vertex for the set of regular graphs given in Table 4.3.

File	**runtime(ms)**	**MTEPs**	**(seq.)x**	**(gunrock)x**	**(ligra)x**
g7j180sc(D)	1.6	467	13.9x	1.7x	1.7x
g7j200sc(D)	1.7	493	14.6x	1.7x	1.8x
mark3j140sc(D)	5.3	76	13.2x	2.1x	1.2x
smallworld(U)	1.0	1000	27.6x	1.5x	1.5x
ASIC-100ks(D)	2.7	215	25.7x	1.6x	1.7x
ASIC-680ks(D)	6.6	353	43.9x	1.0x	1.5x
com-Youtube(U)	9.7	616	48.4x	1.0x	2.8x
mawi-12345(U)	74.8	509	33.6x	1.0x	3.6x
mawi-20000(U)	143.0	521	33.9x	1.0x	3.4x
mawi-20030(U)	261.4	549	32.3x	1.0x	3.2x

the best performance. This algorithm obtained up to 1000 MTEPs, a maximum of 48.4x and an average of 28.7x speedup over the sequential code, a maximum of 2.1x and average of 1.3x speedup over the BC algorithm available in the gunrock library, and a maximum of 3.6x and an average of 2.2x speedup over the BC algorithm available in the ligra library. The scale-free metrics *scf* for this group of regular graphs was in the range [2,224] and 80 % of the graphs have a value less than 100 for this metric.

Our experiments also showed that for the last four graphs in Table 4.4 with vertices with a maximum degree much higher than the mean value, the TurboBC-scCOOC based on the COOC format had a better performance than the TurboBC algorithms based on the CSC format, and also than the corresponding algorithms in the ligra library, asserting that the COOC format results in scalar algorithms that are less affected for vertices with high degrees as compared to scalar algorithms based on the CSC format.

4.4.2 Experimental results for irregular graphs

This section summarizes the results of the experiments performed to compute the BC of one vertex on the nine irregular undirected (U) graphs. The number of vertices (n) and edges (m), the parameters (maximum, mean, standard deviation) of the degree distribution, the depth of the BFS tree (d), and the scale-free metrics (scf), are given for each graph in Table 4.5, and Table 4.6 includes the runtime, MTPEs and the speedup obtained by the TurboBC algorithms over the algorithms implemented on the gunrock (**(gunrock)x**) and ligra (**(ligra)x**) libraries, and over the sequential algorithm (**(seq.)x**).

As expected, the scale-free metrics *scf* for the irregular graphs given in Table 4.5, varied in a wider range [5846,651837], than those values of *scf* obtained for regular graphs given in Table 4.1. The best performance on the computation of BC

Table 4.5: Parameters for the set of directed (D) and undirected (U) irregular graphs with experimental results given in Table 4.6.

File	**n**$\times 10^3$	**m**$\times 10^3$	**degree(max**$/\mu/\sigma$)	**d**	**scf**
mycielski15(U)	25	11111	12287/452/664	3	41166
mycielski16(U)	49	33383	24575/679/1078	3	82833
mycielski17(U)	98	100246	49151/1020/1747	3	166407
mycielski18(U)	197	300934	98303/1531/2817	3	333199
mycielski19(U)	393	903195	196607/2297/4530	3	651837
kron-logn18(U)	262	21166	49164/81/454	6	5846
kron-logn19(U)	524	43563	80676/83/541	6	6609
kron-logn20(U)	1049	89241	131505/85/641	6	7410
kron-logn21(U)	2097	182084	213906/87/756	6	8161

Table 4.6: Experimental runtime, MTPEs and the speedup obtained by the TurboBC-veCSC algorithm over the algorithms implemented on the gunrock (**(gunrock)x**) and ligra (**(ligra)x**) libraries, and over the sequential algorithm (**(seq.)x**), with the computation of BC/vertex for the set of irregular graphs given in Table 4.5.

File	**runtime(ms)**	**MTEPs**	**(seq.)x**	**(gunrock)x**	**(ligra)x**
mycielski15(U)	1.7	6536	17.4x	1.2x	2.3x
mycielski16(U)	3.4	9819	26.6x	1.5x	3.4x
mycielski17(U)	7.9	12689	34.6x	1.7x	4.4x
mycielski18(U)	18.5	16267	45.8x	2.1x	5.1x
mycielski19(U)	48.9	18470	53.1x	2.7x	5.2x
kron-logn18(U)	8.7	2433	31.6x	0.9x	1.1x
kron-logn19(U)	17.4	2504	44.7x	1.0x	0.9x
kron-logn20(U)	58.4	1528	34.0x	1.3x	1.0x
kron-logn21(U)	193.2	943	24.5x	1.1x	1.0x

for these irregular graphs was obtained by the TurboBC-veCSC algorithm as it was expected. This algorithm obtained up to 18.5 GTEPs, as well as a maximum of 53.1x and an average of 34.7x speedup over the sequential code, a maximum of 2.7x and an average of 1.5x speedup over the BC algorithm available in the gunrock library, and a maximum of 5.2x and an average of 2.7x speedup over the BC algorithm available in the ligra library. The TurboBC-veCSC algorithm obtained the maximum values of speedup and MTEPs for the mycielski group of graphs for which the depth (d) of the BFS tree was equal to 3, and a scale-free metrics above 41000.

4.4.3 Experimental results for big graphs

This section summarizes the results of the experiments performed to compute the BC of one vertex on four big graphs. The first graph in Table 4.7 is a regular graph for which the TurboBC-scCSC algorithm showed the best performance, and the other three graphs are irregular directed graphs. The directed graph sk-2005 in Table 4.7 is the largest graph for which the TurboBC was computed with our available GPU. The number of vertices (n) and edges (m), the parameters (maximum, mean, standard deviation) of the degree distribution, the depth of the BFS tree (d), and the scale-free metrics (scf), are given for each graph in Table 4.7, and Table 4.8 includes the runtime, MTPEs and the speedup obtained by the TurboBC algorithms over the algorithms implemented on the ligra (**(ligra)x**) library, and over the sequential algorithm (**(seq.)x**).

Table 4.7: Parameters for the set of directed (D) and undirected (U) big graphs with experimental results given in Table 4.8.

File	$\mathbf{n} \times 10^6$	$\mathbf{m} \times 10^6$	**degree**($\mathbf{max}/\mu/\sigma$)	**d**	**scf**
kmer-V1r(U)	214	465	8/2/1	324	2
it-2004(D)	42	1151	9964/28/67	50	543
GAP-twitter(D)	62	1469	3×10^6/24/1990	15	126
sk-2005(D)	51	1950	12870/39/78	54	1262

Table 4.8: Experimental runtime, MTPEs and the speedup obtained by the TurboBC-veCSC algorithm over the algorithms implemented on the ligra (**(ligra)x**) library, and over the sequential algorithm (**(seq.)x**), with the computation of BC/vertex for the set of big graphs given in Table 4.7. The BC algorithm implemented on the gunrock library ran out of memory for these big graphs.

File	**runtime(s)**	**MTEPs**	**(seq.)x**	**(ligra)x**
kmer-V1r(U)	14.3	33	94.5	0.9x
it-2004(D)	3.1	371	39.5	0.8x
GAP-twitter(D)	7.3	201	50.4	0.8x
sk-2005(D)	6.8	287	30.5	0.7x

For the it-2004 irregular graph, the best performance was obtained with the TurboBC-scCOOC algorithm, for the other two irregular graphs the best performance was obtained with the TurboBC-veCSC algorithm, because the TurboBC-scCOOC algorithm ran out of memory (OOM). For all the graphs on the set, the BC algorithm on the gunrock library ran out of memory (OOM), asserting our optimization strategy of reducing the memory footprint to design and implement our highly scalable TurboBC algorithms.

The TurboBC algorithms obtained for this BC computation for big graphs, up to 371 MTEPs, and a maximum of 94.5x and an average of 53.8x speedup over the sequential code. Since the BC algorithms in the ligra library used the CPU resources, especially the memory resources, effectively, there were up to 1.4x faster than the TurboBC algorithms for the computation of BC in these big graphs.

Figure 4.3a) shows that the greatest speedups of the TurboBC algorithms over the sequential BC algorithm were for the regular graph with the greatest value for the depth (d) of the BFS tree and that the maximum values for the METPs were obtained with the TurboBC-veCSC algorithm applied to the irregular directed graphs, for which the depth (d) of the BFS tree was equal to or less than 50.

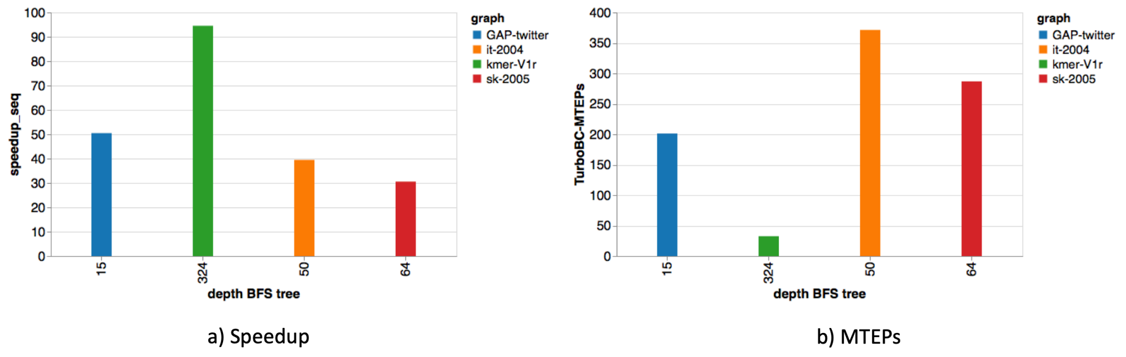


Figure 4.3: Experimental results for a) the speedup over the sequential algorithm and b)MTEPs obtained for our TurboBC algorithms in the computation of BC of the set of big graphs of Table 4.7.

4.4.4 Experimental results for the exact BC computation of all vertices of a graph

Table 4.9 summarizes the experimental results obtained by the TurboBC algorithms for the exact BC computation for all vertices of the set of six graphs. The first four are directed regular graphs, and the last two are undirected irregular graphs. The parameters and the TurboBC algorithm used for these computations for the regular graphs are included in Tables 4.1 and 4.3, and for the irregular graphs in Table 4.5. The parameter $\mathbf{n} \times \mathbf{m}$ is included in Table 4.9, because the MTEPs for the exact BC computation are computed as the ratio between the number of edges times the number of vertices (millions) and the average runtime (seconds).

The TurboBC algorithms obtained for this exact BC computation, up to 13.8 GTEPs and a maximum of 38.0x and an average of 18.4x speedup over the sequential code. The results in Table 4.9, also show that both the speedup and the MTEPs increased with the size of the graph, showing the high scalability of the TurboBC algorithms for this type of computation.

Table 4.9: Experimental runtime, MTEPs, and speedup obtained with the TurboBC algorithms over the sequential algorithm (**(seq.)x**) for the computation of the exact BC of all vertices of a set of undirected and directed graphs.

File	**d**	**n×m** $\times 10^6$	**runtime(s)**	**MTEPs**	**(seq.)x**
mark3j60sc(D)	42	4694	49.3	95	8.2x
mark3j80sc(D)	52	8345	90.8	92	9.2x
g7j180sc(D)	17	39906	105.9	377	13.4x
g7j200sc(D)	17	49688	129.7	383	14.3x
mycielski16(U)	3	1639081	159.8	10257	27.5x
mycielski17(U)	3	9854152	715.2	13778	38.0x

Figure 4.4 shows that the maximum values for speedups and for MTEPs were obtained, for the graphs with the smaller values of the depth (d) of the corresponding BFS trees.

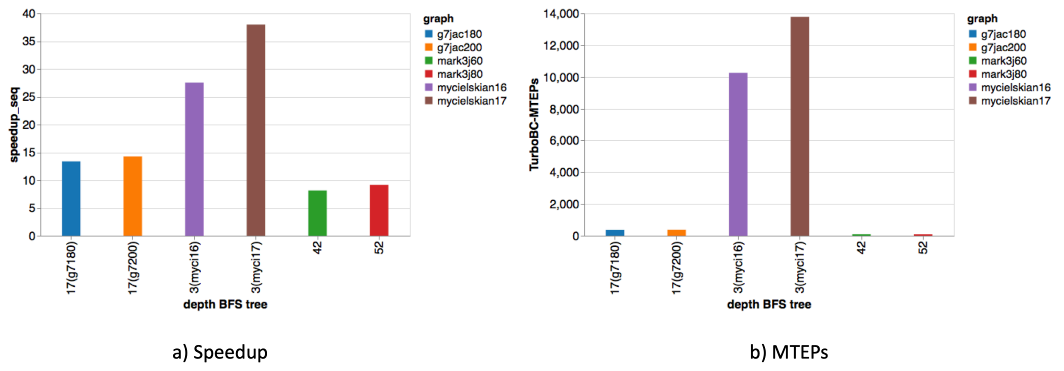


Figure 4.4: Experimental results for a) the speedup and b) the MTEPs of the exact computation of BC for all vertices of the graphs given in Table 4.9.

4.4.5 GPU memory usage by the TurboBC algorithms

Figure 4.5 c) compares the GPU memory usage by the TurboBC-veCSC algorithm and by the BC algorithms in the gunrock library during the computation of BC for the mycielski group of irregular graphs given in Table 4.5. Since the space complexity of the Brandes' algorithm is O(m+n), Figure 4.5 c) shows that there is a linear relationship between the GPU memory usage and the sum of the number of vertices plus the number of edges of the mycielski graphs. Due to the strategy of reducing the memory footprint of the TurboBC algorithms, the memory usage of the gunrock library was up to 60 % higher than the memory usage of the TurboBC-veCSC algorithm.

To quantify the reduction in the memory footprint of the TurboBC algorithms, Figure 4.6 shows the data flow for the BC algorithms in the gunrock library and for our TurboBC algorithms. The host (CPU) arrays (yellow) are shown as inputs to the memory transfer block, the auxiliary arrays (green) are used by the GPU to compute the BC which results in the output arrays (blue). The size of each array is given, with n as the number of vertices and m as the number of edges. We assumed that a lower bound for the global memory required by the GPU during the

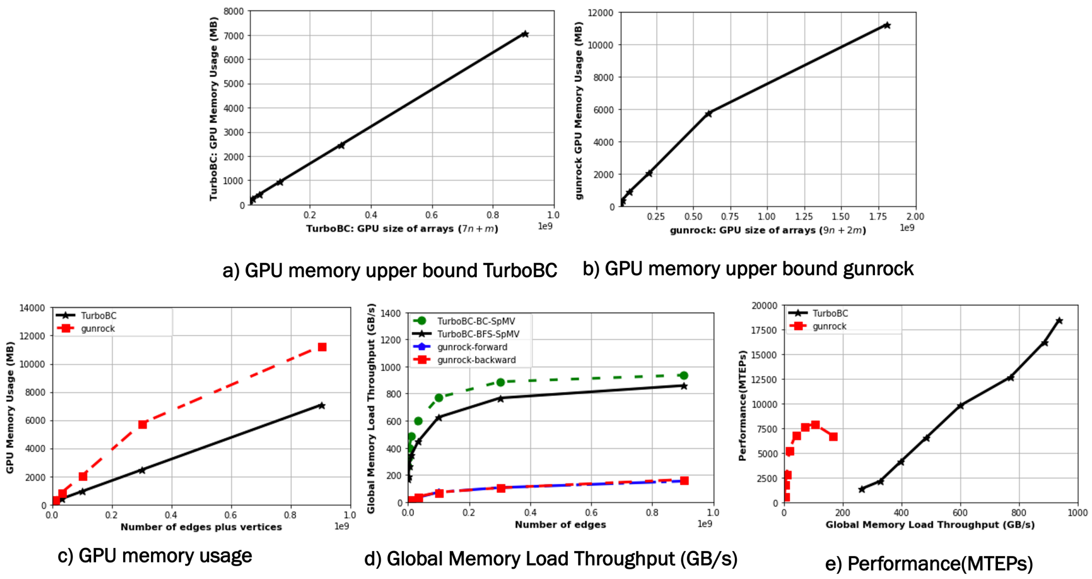


Figure 4.5: GPU memory usage, GPU memory upper bounds, Global Memory Load Throughput (GLT) and performance(MTEPs) obtained with the TurboBC-veCSC algorithm compared with the values obtained by the gunrock BC algorithms for the computation of BC/vertex in the mycielski group of irregular graphs included in Table 4.5.

BC computation was proportional to the total size of the arrays required by this computation, which in the case of the gunrock library is equal to $9n + 2m$, and for the TurboBC is equal to $7n+m$ for the BC computation stage. Figures 4.5 a) and b) show the expected linear relationship between the GPU memory usage, considered an experimental GPU memory upper bound, and the total size of the arrays for the computation of BC with TurboBC and with the gunrock libray, respectively. Our experimental results also showed that the reduction, proportional to $2n + m$, in the GPU global memory requirements of the TurboBC algorithms, illustrated in Figure 4.5 c), allowed the computation of the BC for the relatively big graphs given in Table 4.7, while the BC algorithms in the gunrock library ran out of memory for these big graphs.

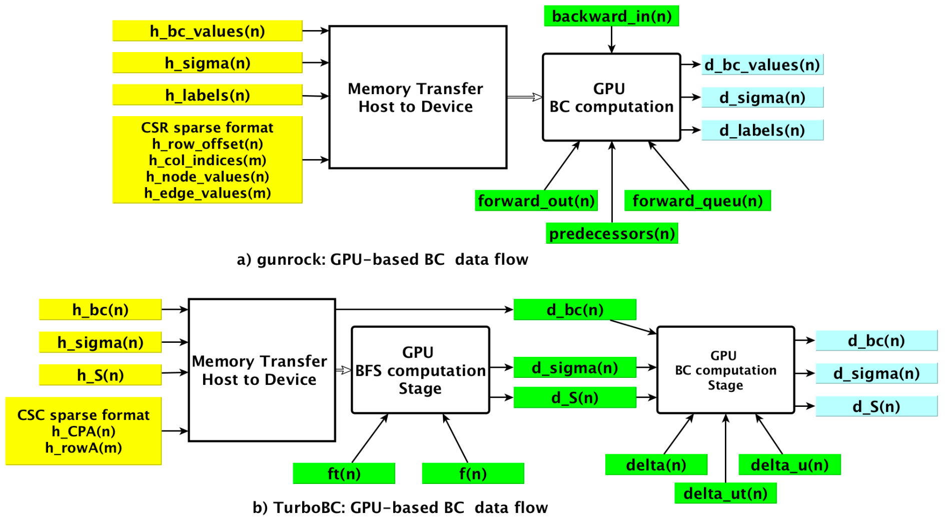


Figure 4.6: Data flow for the GPU-based BC algorithms implemented in the gunrock library and in TurboBC.

GPU Global Memory Load Throughput (GLT) and MTEPS obtained by the TurboBC-veCSC algorithm: The Global Memory Load Throughput (GLT) is a GPU metric that measures the rate at which the GPU global memory is accessed by an SM [cor21], Figure 4.5 d) compares this metric obtained by the most important kernels of the TurboBC-veCSC algorithm, and by the kernels of the BC algorithm in the gunrock library. The theoretical maximum GLT achievable for the GPU (NVIDIA Titan Xp) used in our experiments was 575 GB/s, represented by the horizontal line in Figure 4.5 d), the GLT obtained by the kernels in the gunrock library were substantially below this value, while the kernels in the TurboBC-veCSC algorithm obtained GLT values that were 60 % higher than the theoretical maximum GLT. Figure 4.5 e) illustrates that the MTEPs, as a function of the GLT metric, obtained by the TurboBC-veCSC algorithm were much higher than those obtained by the BC algorithms in the gunrock library.

In summary, the experimental results presented in Figure 4.5, showed that for a representative group of highly irregular big graphs, the TurboBC algorithms used memory more efficiently than the BC algorithms in the gunrock library.

4.5 Summary

Betweenness centrality (BC) is a shortest path centrality metric used to measure the influence of individual vertices or edges on huge graphs that are used for modeling and analysis of the human brain, omics data, or social networks. The application of the BC algorithm to modern graphs must deal with the size of the graphs, as well as highly irregular data-access patterns. These challenges are particularly important when the BC algorithm is implemented on Graphics Processing Units (GPU), due to the limited global memory of these processors, as well as the decrease in performance due to the load unbalance resulting from processing irregular data structures.

In this chapter, as far as we know, we present the first GPU-based linear-algebraic formulation and implementation of BC, called TurboBC, a set of memory-efficient BC algorithms that exhibits good performance and high scalability on unweighted, undirected, or directed sparse graphs of arbitrary structure.

Our extensive set of experiments showed that the TurboBC algorithms obtained more than 18 GTEPs (billions of transverse edges per second), and an average speedup of 31.9x over the sequential version of the BC algorithm, and were on average 1.7x and 2.2x faster than the state-of-the-art algorithms implemented on the high performance, GPU-based, gunrock [WDP$^+$16], and CPU-based, ligra [SB13] libraries, respectively. These experiments also showed that by minimizing their memory footprint, the GPU memory usage of the gunrock library was higher than the memory usage of the TurboBC algorithms, allowing these algorithms to compute the BC of relatively big graphs, for which the gunrock algorithms ran out of memory. Our experiments also demonstrated that the performance obtained by the TurboBC algorithms, measured as MTEPs, as a function of the GPU memory bandwidth, were much greater than those obtained by the BC algorithms in the gunrock library, showing that the GPU memory was used more efficiently by the TurboBC algorithms. The codes to implement the algorithms proposed in this chapter are available at https://github.com/pcdslab/TurboBC.

Our future work will be focused on improving the performance of the algorithms in TurboBC, especially the performance of the algorithms computing the vector sparse matrix multiplication operations. Our goal will be to design and implement memory-efficient, on which the usage of the shared memory in the GPU will be optimized, and scalable GPU-based BC algorithms, with higher performance for big graphs than the state-of-the-art BC algorithms.

CHAPTER 5

CONFOUNDING EFFECTS ON THE PERFORMANCE OF MACHINE LEARNING ANALYSIS OF STATIC FUNCTIONAL CONNECTIVITY COMPUTED FROM RS-FMRI MULTI-SITE DATA

In this chapter, we present a comprehensive approach for the solution of the problem of confounding effects over the machine learning classification models of rs-fMRI multisite-data [AAMS23].

5.1 Introduction

Resting-state functional magnetic resonance imaging (rs-fMRI) is a non-invasive imaging technique based on the blood oxygen level of the brain [OLNG90, OMT$^+$93], widely used in neuroscience to understand the functional connectivity of the human brain. An active area of research in neuroscience is the modeling of rs-fMRI data, using complex graph theory, to discover the functions and structure of the human brain, and for the detection of brain disorders [SCKH04, STK05, SR07, vdHSBP08, BB11, Spo12, BS17].

Initial fMRI studies based on data collected in a single imaging site, usually had limited statistical power, due to the difficulties in obtaining large amounts of data such as the limited participants with brain disorders in one geographical location, as well as limited resources [VHT09]. To overcome these limitations, multi-site neuroimaging data have been extensively used in network neuroscience research in the last decade [FGC$^+$06, FSB$^+$08, VHT09, BMZ$^+$10, GGW$^+$10, PBG$^+$12, NSF$^+$17, RMMM$^+$17]. The Autism Brain Imaging Data Exchange (ABIDE) functional magnetic resonance database [CBC$^+$13, DMYL$^+$14, DMOC$^+$17] exemplifies a modern multi-site rs-fMRI database which provides a larger sample size of rs-fMRI data obtained from a more heterogeneous population living in different geographical locations, resulting in higher statistical power compared to the rs-fMRI data obtained for a single site [VHT09, BMZ+10]. The ABIDE database is a powerful tool for enhancing the reproducibility and the reliability of the statistical methods and models implemented for the diagnosis and discovery of autism spectrum disorders [AMDM+17, EMF+19, AS21a].

One main challenge for the neuroscience research community using rs-fMRI multi-site databases is the existence of confounding effects, associated with variables resulting from imaging and population heterogeneity among different sites. Several studies have shown that these confounding factors affect the performance of the machine learning models when executed on rs-fMRI multi-site data [PBM15, KFMB+16, AMDM+17]. One main effect is the increase in variability, as well as the imposition of upper limits on the classification scores, due to the decrease of statistical power of the machine learning classification of patients and control subjects.

A first group of confounding effects are those resulting from the imaging acquisition such as MRI scanner vendor, scanner technology, magnetic field strength and inhomogeneities, and scanning protocols and parameters for the image acquisition, such as scan length, repetition time, echo time, acquisition time, and voxel size [FGC+06, FSB+08, GJM+10, BMS+11, BMP+13, KAD+14, CLC+14, FMG+14, FSF+15, MNS+16, AMDM+17]. The control and reduction of these imaging confound effects have been partially solved by implementing standard protocols and parameters for the image acquisition procedures [FSB+08, GMT+12, SON+17, CVZ+18].

A second group of confounding effects are those related to phenotypic data derived from the heterogeneous population from which the MRI data is obtained, i.e., clinical information of patients (e.g., taking medications, severity of disorder symptoms), instructions given to the subjects during testing (e.g., eyes open or closed), as well as relevant demographic data (e.g., age range, IQ-range, gender) [VHT09, DSMI11, BMP^{+}13, CLC^{+}14, VS13, AMR^{+}17, RMMM^{+}17, DBR^{+}17, FCS^{+}18, BCVO^{+}20, RLH21, RJF^{+}21, BPP^{+}22]. Some studies have implemented stratification techniques [Par14] of the rs-fMRI data of the ABIDE sites to control the confounding effects due to diverse phenotypic data. These stratification techniques were used to generate sub-samples integrated by subjects sharing common characteristics such as gender, age, right-handed, and eyes open, to obtain more homogeneous and suitable data sets for the statistical analysis of the static functional connectivity derived from rs-fMRI multi-site data [CKM13, NZF^{+}13, VMSS13, CKJ^{+}15, PBM15, Iid15, KFMB^{+}16, AMDM^{+}17, GDM^{+}17, KSL17, SKB^{+}17, PKF^{+}18, WXW19, KGX^{+}19, KJKS19, LGD^{+}20, SAS^{+}20].

During the last decade, important research efforts have been dedicated to identifying the confounding variables and controlling the corresponding effects over the statistical analysis of multi-site MRI data. Diverse studies implemented statistical regression models to quantify and control the confounding effects over predictive modeling using multi-site structural MRI data [RMMM^{+}17], as well as rs-fMRI data [DBR^{+}17]. The harmonization models, also known as combined batch (ComBat) harmonization models, are based on an empirical Bayes model, originally proposed to control batch effects introduced by different samples in gene expression microarrays experiments by Johnson et. al. [JLR07]. This model was reformulated in the context of heterogeneous multi-site diffusion tensor imaging data by Fortin et. al. [FPT^{+}17], to remove confounding effects introduced by the technical differences of the scanners used by the different sites, while conserving the variability introduced by selected phenotypic variables. Some studies also implemented the ComBat harmonization models to correct site effects in the statistical analysis of static functional connectivity computed from multi-site rs-fMRI data [YLC$^+$18, YYI$^+$19, RLH21, TMA$^+$21, CSP$^+$22].

In this study, we used the ABIDE rs-fMRI data with the 17 international imaging sites summarized in Table 5.1. The goals of our study were twofold i) the identification of the phenotypic and imaging variables producing the confounding effects, and ii) to control these confounding effects to maximize the classification scores obtained from the machine learning analysis the rs-fMRI ABIDE multi-site data. To achieve these goals, we proposed two sets of methods. The first set of methods was implemented to generate new features for the machine learning models. These new features were computed from the static functional connectivity values computed from the rs-fMRI multi-site data. The first methods implemented in this set were multiple linear regression (MLR) models mainly applicable to the identification of the confounding variables, however, the experimental results showed that they were also useful in maximizing the classification scores computed with the machine learning models (see Section 5.2.5). The second method implemented in this set was ComBat harmonization models implemented to control the confounding effects and to maximize the classification scores (see Section 5.2.6). Since the independent variables of the MLR and ComBat harmonization models give only a partial explanation of the variability of the dependent variables, we also generated new features by using normalization methods on which the confound variables were unknown (see Section 5.2.7). The second set of methods was based in the stratification techniques defined by [Par14] and [Ney92] which basically consists of probability sampling methods on which the subjects of the target population are divided into sub-samples or strata where within each sub-sample the subjects have similar characteristics. These techniques were implemented to generate homogeneous sub-samples of the 17 ABIDE sites on which the subjects were in different ranges of age and/or full IQ (FIQ) (see Section 5.2.8).

The main contribution of the work presented in this chapter is a comprehensive approach to the solution of the problem of confounding effects over the machine learning classification models of rs-fMRI multisite data, consisting of the sets of proposed methods as well as the extensive set of experiments performed with these methods. The experimental results were also thoroughly analyzed and compared to evaluate the effectiveness of each one of the implemented methods. The proposed approach can be used and improved by the neuroscience research community to help in the diagnosis of brain disorders.

Table 5.1: International Imaging Sites from preprocessed ABIDE resting state fMRI data (http://preprocessed-connectomes-project.org/abide/) used in this chapter [CBC+13].

Sites: California Institute of Technology (Caltech), Carnegie Mellon University (CMU), Kennedy Krieger Institute (KKI), University of Leuven (Leuven), Ludwig Maximilian University (MaxMun), Oregon Health and Science University (OHSU), Institute of Living at Hartford Hospital (Olin), University of Pittsburgh School of Medicine (Pitt), Social Brain Lab (SBL), San Diego State University (SDSU), Stanford University (Stanford), Trinity Center for Health Sciences (Trinity), University California Los Angeles (UCLA),University of Michigan (UM), University of Utah School of Medicine (USM), and Child Study Center, Yale University (Yale).

MRI vendors: General Electric (GE), Phillips(P), Siemens(S)

Site	C	ASD	Subjects	Avg age	Avg FIQ	M/F	MRI
Caltech	18	19	37	27.7 ± 10.3	111.5 ± 11.2	29/8	S
CMU	13	14	27	26.6 ± 5.6	114.6 ± 10.3	21/6	S
KKI	28	20	48	10.0 ± 1.3	106.2 ± 4.8	36/12	P
Leuven	34	29	63	18.0 ± 5.0	107.6 ± 18.0	55/8	P
MaxMun	28	24	52	25.3 ± 11.8	110.9 ± 11.4	48/4	S
NYU	100	75	175	15.3 ± 6.5	110.5 ± 14.9	139/36	S
OHSU	14	12	26	10.7 ± 1.8	111.0 ± 16.3	26/0	S
Olin	15	19	34	16.6 ± 3.4	113.2 ± 16.5	29/5	S
Pitt	27	29	56	18.9 ± 6.9	110.2 ± 12.1	48/8	S
SBL	15	15	30	34.4 ± 8.5	107.9 ± 9.4	30/0	P
SDSU	22	14	36	14.4 ± 1.8	109.4 ± 13.6	29/7	GE
Stanford	20	19	39	10.0 ± 1.6	111.4 ± 15.4	31/8	GE
Trinity	25	22	47	17.0 ± 3.4	110.0 ± 13.6	47/0	P
UCLA	44	54	98	13.0 ± 2.2	103.1 ± 12.7	86/12	S
UM	74	66	140	14.0 ± 3.2	106.9 ± 13.6	113/27	GE
USM	25	46	71	22.7 ± 8.3	105.2 ± 17.5	71/0	S
Yale	28	28	56	12.7 ± 2.9	99.8 ± 19.9	40/16	S
TOTAL	530	505	1035			878/157	

5.2 Methods and materials

5.2.1 ABIDE resting fMRI multi-site data

Functional magnetic resonance imaging (fMRI) is based on the fact that hemoglobin, the carrier of oxygen from the lungs to the tissues [MR06], changes its magnetic properties depending on its level of oxygenation. The neuronal activity of the brain requires energy, which is supplied by glucose, and oxygen transported by hemoglobin. Hence, in regions of intense neuronal activity, the oxygen carried by the hemoglobin molecules is consumed, resulting in changes in the magnetic properties of these molecules. Such oxygen dependence makes hemoglobin a sensitive magnetic marker of the level of blood oxygenation, and consequently of neuronal activity. The different interactions of the oxygenated and deoxygenated hemoglobin with magnetic fields can be detected as changes in the fMRI signal. This relationship between neuronal activity in different regions of the brain and the fMRI signal is known as the blood oxygenation level-dependent (BOLD) effect, and the resulting functional imaging is known as BOLD fMRI. Since the rs-fMRI time series, recorded from subjects who are at rest at the scanner, reflect dynamic changes in the brain due to neuronal activity in different regions of the brain, they can be used to estimate the functional connectivity between these regions [AGHP89, BZYHH95, vdVFP+04].

The rs-fMRI measured with the MRI scanners need to be preprocessed to correct for confounding effects such as magnetic field distortions and head motion, as well as to improve the signal-to-noise ratio [JC18]. The preprocessed rs-fMRI data used in this study was obtained from the 17 international imaging sites listed in Table 5.1, publicly available in the ABIDE database, with a total of 530 control and 505 autism subjects [CBC+13, DMYL+14, DMOC+17]. The preprocessing pipeline chosen for this data was the Configurable Pipeline for the Analysis of Connectomes (CPAC), and the filt-global preprocessing strategy, on which the head motion correction is performed using a two-stage approach as described in https://fcp-indi.github.io/docs/latest/user/quick.html and [CJ99]. The preprocessing pipeline is described in detail in the ABIDE Preprocessed website (http://preprocessed-connectomes-project.org/abide/index.html).

5.2.2 Human brain functional networks

In the last two decades, the graph theoretical analysis of functional connectivity between brain regions, on which the rs-fMRI data is represented as human brain functional networks, has been fundamental to identifying organizational principles in the brain, as well to understanding the causes of brain disorders [SCKH04, STK05, SR07, vdHSBP08, BB11, Spo12, BS17].

In this work, the human brain functional networks, which will be referred to as functional networks for the rest of the chapter, were represented as weighted, undirected graphs $G = (V, E)$, where V is the finite set of nodes and E the set of edges. Any pair $(u, v) \in E$ implies that the vertices u and v in V are connected by an edge in G, with the weights of the edges equal to the values of the static functional connectivity between the regions of the brain represented by the corresponding nodes.

Nodes of the functional network

The first and most critical step in the construction of functional networks is the definition of the network nodes. The voxel-based and the brain parcellation are two of the most used methods for this definition. In the first method, the voxels, small 3D volumes of the MRI images, are used to define the nodes of the functional networks. One important limitation of this approach is that the voxels subdivided the brain into regions that do not represent any biological structure. Furthermore, since there are tens of thousands of voxels, the corresponding networks are very noisy and with very high dimensionality, yielding functional connectivity features that are unusable for an efficient machine learning analysis of the functional connectivity and graph metrics derived from the rs-fMRI data.

The brain parcellation approach provides important information about the functional and anatomical organization of the human brain by subdividing the brain into a set of distinct and coherent regions of interest (ROI), where each ROI is formed by a group of voxels with biological meaning. Each ROI corresponds to a node of the functional network, and the rs-fMRI time series of each ROI is computed as the average of the time series of all the voxels grouped in the region. The static functional connectivity values are derived from these average rs-fMRI time series. Hence, the computation of rs-fMRI time series at the ROI-level, results in a biologically informed type of feature extraction, compressing the high dimensionality of the time series of hundreds of thousands of voxels to the lower dimensionality of the time series computed for the set of ROIS. These feature extraction procedures are not only important to avoid over-fitting of the machine learning models used for the analysis of rs-fMRI data, but also by providing a limited set of nodes for the application of the mathematical tools of graph theory to the modeling and analysis of brain connectivity.

In this study, we selected the brain parcellation, referred to as the brain atlas for the rest of the chapter, for which the highest values of classification scores were obtained from the machine learning analysis of the static functional connectivity obtained from the average rs-fMRI time series of the 17 ABIDE sites given in Table 5.1. We preselected three of the brain atlases provided by the ABIDE database. The first one was the cc200 (200 nodes) brain atlas, which was derived from rs-fMRI data [CJHI+12]. The other two were the Automated Anatomical Labeling (aal, 116 nodes) [TMLP+02], and the Harvard-Oxford (ho, 111 nodes) [ESM+05] brain atlases, both derived from structural anatomic information. Additional information about these brain atlases is given in [EJCB12, YHX+15, AKM+18, Mes20]. We computed the classification scores for each ABIDE site, using the values of functional connectivity as features of ASD-DiagNet (see Section 5.2.3). The computed classification scores showed that the cc200 brain atlas obtained the highest values of the classification scores for most of the ABIDE sites, hence this atlas was selected to represent the nodes of the functional networks from which all the experimental results presented in this chapter were obtained [AS21b].

Edges of the functional network

In this study, the weights of edges, i.e., the elements of the static functional connectivity adjacency matrix of the functional network, were obtained by computing the linear correlation between the time series for all pairs of nodes, using the Pearson correlation function available in the NumPy package (https://numpy.org) and given by

$$FC_{uv} = \frac{\sum_{t=1}^{T}(tu_t - \hat{tu})(tv_t - \hat{tv})}{\sqrt{\sum_{t=1}^{T}(tu_t - \hat{tu})^2}\sqrt{\sum_{t=1}^{T}(tv_t - \hat{tv})^2}} \tag{5.1}$$

were tu and tv are the rs-fMRI time series, with length T, of the nodes u and v of the cc200 brain atlas, and $\hat{tu}$ and $\hat{tv}$ the averages of these time series. Since the static functional connectivity adjacency matrix is symmetric, and the diagonal elements contain no relevant information, a common feature selection method is to select the upper (or the lower) triangular part of this matrix, which are referred to as functional connectivity ($\boldsymbol{FC}$) for the rest of the chapter, reducing the size of the features from N^2 to $N(N-1)/2$, where N is the number of ROIs of the brain atlas.

For the cc200 atlas, the number of nodes is $N = 200$, then, the number of values of the functional connectivity is reduced from $40,000$ to $19,990$, less than half the original size.

5.2.3 The machine learning models: ASD-DiagNet and ASD-SAENet

For this study, we selected two state-of-the-art machine learning models: ASD-DiagNet and ASD-SAENet to perform the experiments of classification of control and autistic subjects, and to compare the corresponding results.

The classification scores computed in our experiments were: *Accuracy* which measures the ratio of correctly classified patient subjects (true positive) and control subjects (true negative) over the total number of subjects; *sensitivity* which measures the ratio of the correctly classified as patient subjects over the total number of patients (true positive plus false negative); and *specificity* which measures the ratio of the correctly classified as control subjects over the total number of control subjects (true negative plus false positive), more details about these scores are given in [ZZW+10].

ASD-DiagNet

ASD-Diagnet was selected as one of the machine learning classifiers to compute the experimental results included in this study. ASD-DiagNet is a GPU-based machine learning model for classifying patients and control subjects by using only rs-fMRI data. ASD-DiagNet was designed to implement a joint learning procedure using an autoencoder for feature extraction, i.e., to compress the original feature space into a lower dimensional space that contains useful patterns of the original data.

The lower dimensional data generated by the autoencoder was used as input for the classification step performed by a single-layer perceptron (SLP) classifier. The features selected for the training samples of ASD-DiagNet were 25 % of the maximum weights and the same percentage of the minimum weights of the functional connectivity values. For all our experiments, we used the data augmentation method using linear interpolation implemented for ASD-DiagNet. A detailed description of ASD-DiagNet is given in [EMF+19].

ASD-SAENet

ASD-SAENet was the other machine learning classifier used to perform a selected set of experiments to compare their results with those computed with ASD-DiagNet. ASD-SAENet is a GPU-based machine learning model for classifying patients and control subjects by using only rs-fMRI data. ASD-SAENet was designed and implemented as a sparse autoencoder (SAE) which results in optimized extraction of features that can be used for classification. These features are then fed into a deep neural network (DNN) to perform the classification of control and autistic subjects. This model is trained to optimize the classifier while improving extracted features based on both the reconstructed data error and the classifier error. The features selected for the training samples of ASD-SAENet were 25 % of the maximum weights and the same percentage of the minimum weights of the functional connectivity values. ASD-SAENet did not implement data augmentation to minimize overfitting. A detailed description of ASD-DiagNet is given in [AS21a].

5.2.4 Generation of new features

We implemented a set of methods to generate new features for the machine learning models. These new features were computed from the functional connectivity values obtained from the rs-fMRI time series (see Section 5.2.2). The first two methods were multiple linear regression models, and ComBat harmonization models, which were implemented assuming that the variables responsible for the confounding effects were known such as MRI scanner vendor, as well as some phenotypic variables like age, FIQ, and gender. In the third group of methods included in this set, the new features were obtained from normalization methods, for which we assumed that the variables responsible for the confounding effects were unknown. Figure 5.1 illustrates the workflow implemented in this study to generate the new features.

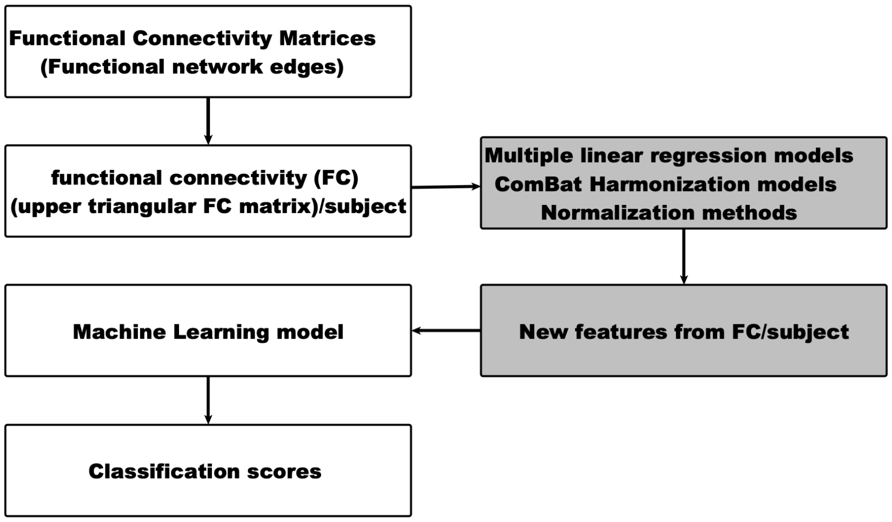

Functional Connectivity Matrices
(Functional network edges)
functional connectivity (FC)
(upper triangular FC matrix)/subject
Multiple linear regression models
ComBat Harmonization models
Normalization methods
New features from FC/subject
Machine Learning model
Classification scores

Figure 5.1: Workflow for the machine learning analysis of rs-fMRI data using new features derived from the functional connectivity values to control the confounding effects of multi-site rs-fMRI data.

A more detailed example of the computation of new features is illustrated by the workflow of Figure 5.2, where the MLR models are included as an example. The functional connectivity values as well as the phenotypic values of the ABIDE subjects were the input data for the creation of a dictionary for each feature with the values of the functional connectivity, subject ID, age, gender, FIQ and MRI vendor of each subject. Then a list of dictionaries was obtained that was used to compute the new features. Algorithm 5.1 shows the details of the computation of this list of dictionaries.

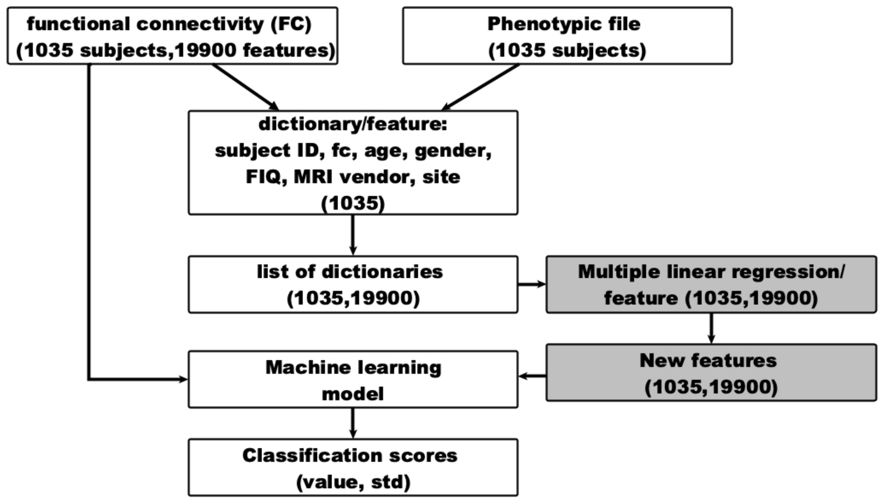

functional connectivity (FC) (1035 subjects,19900 features)
Phenotypic file (1035 subjects)
dictionary/feature: subject ID, fc, age, gender, FIQ, MRI vendor, site (1035)
list of dictionaries (1035,19900)
Multiple linear regression/ feature (1035,19900)
New features (1035,19900)
Machine learning model
Classification scores (value, std)

Figure 5.2: Workflow for computation of new features for the machine learning analysis of the ABIDE rs-fMRI data using the MLR models.

5.2.5 Multiple linear regression models

Multiple linear regression (MLR) models are fitted to random dependent variables $\boldsymbol{Y} = (Y_1, Y_2, ..., Y_n)$, with corresponding observation values $\boldsymbol{y} = (y_1, y_2, ..., y_n)$, to remove the variance that can be explained by the independent or predictor variables.

Algorithm 5.1 Algorithm for the computation of dictionaries for each value of the functional connectivity, and the age, gender, and FIQ of the corresponding subject, as well as the MRI vendor. These dictionaries are appended to a list of dictionaries whose elements are used to compute the new features for the machine learning models described in Section 5.2.3.

```
 1: Input: $\boldsymbol{FC}$                                  ▷ $\boldsymbol{FC}$ : FC values for all the subjects
 2: Input: Subjects                                 ▷ Subjects : list of ABIDE subjects
 3: Output: $\mathbf{FC_{ld}}$                                ▷ $\mathbf{FC_{ld}}$ : List of dictionaries
 4: procedure FCDICT(Subjects, $\boldsymbol{FC}$)
 5:     $\mathbf{m} \leftarrow 19900$                       ▷ m: number of features for cc200 brain atlas
 6:     for $k \to 1, m$ do
 7:         fc ← []
 8:         age ← []
 9:         gender ← []
10:         FIQ ← []
11:         MRI ← []
12:         sub ← []
13:         for $SUB \in Subjects$ do
14:             fc.append ← $\boldsymbol{FC}(SUB)(k)$
15:             age.append ← age($SUB$)
16:             gender.append ← gender($SUB$)
17:             MRI.append ← MRI($SUB$)
18:             sub.append ← SUB
19:         end for
20:         $\mathbf{fc_s}$ ← 'fc' : fc, 'age' : age, 'gender' : gender, 'MRI' : MRI, 'sub' : sub
21:         $\mathbf{FC_{ld}}$.append ← $\mathbf{fc_s}$
22:     end for
23:     return $\mathbf{FC_{ld}}$
24: end procedure
```

The MLR model is given by

$$\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon} \tag{5.2}$$

where $\boldsymbol{X}$ is the design matrix of independent variables, $\boldsymbol{\beta}$ is a vector of unknown parameters and $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, ..., \epsilon_n)$ a vector of random errors with $E(\epsilon_i) = 0$. If the inverse of the matrix $\mathbf{X'X}$ exists, then the ordinary least square (OLS) estimates of the fitted value vector, $\hat{\boldsymbol{y}}$, are given by

$$\hat{\boldsymbol{y}} = \boldsymbol{X(X'X)^{-1}X'y} \tag{5.3}$$

and the residual vector, $\boldsymbol{\Delta y}$, is obtained by removing the variance introduced by the independent variables, represented by the fitted value vector, $\boldsymbol{\hat{y}}$, from the observation values, $\boldsymbol{y}$, of the dependent variables [TD00]

$$\boldsymbol{\Delta y} = \boldsymbol{y} - \boldsymbol{\hat{y}} \tag{5.4}$$

We implemented the multiple linear regression (MLR) models given by Equations 5.2 to 5.4 to quantify the confounding effects of each of the independent variables: age, FIQ, gender, and MRI vendor, as well as the effects of some combinations of these variables. We obtained two sets of new features using the MLR models. The first set was obtained using functional connectivity as the dependent variable, and the second set using the Fisher z-transformation of the functional connectivity, $\mathbf{FC}_{FZ}$ (see Section 5.2.7), as the dependent variable. The fitted values are given by Equations 5.5 and 5.6, and the new features by Equations 5.7 and 5.8, respectively.

$$\widehat{\boldsymbol{FC}} = (\boldsymbol{X}(\boldsymbol{X'X})^{-1}\boldsymbol{X})\boldsymbol{FC} \tag{5.5}$$

$$\widehat{\boldsymbol{FC_{FZ}}} = (\boldsymbol{X}(\boldsymbol{X'X})^{-1}\boldsymbol{X})\boldsymbol{FC_{FZ}} \tag{5.6}$$

$$\Delta \boldsymbol{mlrX} = \boldsymbol{FC} - \widehat{\boldsymbol{FC}} \tag{5.7}$$

$$\Delta \boldsymbol{mlrX}_{FZ} = \boldsymbol{FC}_{FZ} - \widehat{\boldsymbol{FC}}_{FZ} \tag{5.8}$$

The complete set of new features computed with the MLR model, and the corresponding independent variables are given in Table 5.2.

Algorithm 5.2 gives the steps to implement the multiple linear regression of the functional connectivity values to compute the new feature $\Delta \boldsymbol{mlrAGM}$.

Algorithm 5.2 Algorithm to implement multiple linear regression of the functional connectivity values as dependent variables, and age, gender and MRI vendor as independent variables, to compute the new feature $\boldsymbol{\Delta mlrAGM}$, for the machine learning models (Section 5.2.3). The variables and functions are from the statsmodels library (https://www.statsmodels.org/stable/index.html).

```
 1: Input: $\mathbf{FC_{ld}}$                              ▷ $\mathbf{FC_{ld}}$ : FC list of dictionaries
 2: Output: $\boldsymbol{\Delta mlrAGM}$                   ▷ $\boldsymbol{\Delta mlrAGM}$ : New Feature
 3: procedure MLR($\mathbf{FC_{ld}}$)
 4:     $\mathbf{m} \leftarrow 19900$              ▷ $\mathbf{m}$: number of features for cc200 brain atlas
 5:     for $k \rightarrow 1, m$ do
 6:         $\boldsymbol{y, X} \leftarrow \text{dmatrices}(fc \sim \text{age} + \text{gender} + \text{MRI}, \text{data} = \mathbf{FC_{ld}}(k))$
 7:         model ← sm.OLS($\boldsymbol{y, X}$)
 8:         results ← model.fit
 9:         $\boldsymbol{\Delta mlrAGM}$ ← results.residuals
10:     end for
11:     return $\boldsymbol{\Delta mlrAGM}$
12: end procedure
```

Table 5.2: New features computed with the multiple linear regression models and the ComBat harmonization models described in Sections 5.2.5 and 5.2.6.

MLR features	ComBat features	Independent variables
$\boldsymbol{\Delta mlrA}$, $\boldsymbol{\Delta mlrA}_{FZ}$	$\boldsymbol{cbA}$, $\boldsymbol{cbA}_{FZ}$	**age**
$\boldsymbol{\Delta mlrF}$, $\boldsymbol{\Delta mlrF}_{FZ}$	$\boldsymbol{cbF}$, $\boldsymbol{cbF}_{FZ}$	**FIQ**
$\boldsymbol{\Delta mlrG}$, $\boldsymbol{\Delta mlrG}_{FZ}$	——-	**gender**
$\boldsymbol{\Delta mlrM}$, $\boldsymbol{\Delta mlrM}_{FZ}$	——-	**MRI vendor**
$\boldsymbol{\Delta mlrAGM}$, $\boldsymbol{\Delta mlrAGM}_{FZ}$	——-	**age, gender, MRI vendor**
——-	$\boldsymbol{cbAFG}$, $\boldsymbol{cbAFG}_{FZ}$	**age, FIQ, gender**

5.2.6 ComBat harmonization models

In addition to the multiple linear regression models, we implemented the ComBat harmonization models [JLR07, FPT+17, FCS+18, YLC+18], to remove confounding effects introduced by the technical differences of the scanners used by the different sites, while conserving the variability introduced by selected phenotypic and MRI vendors variables, and to determine which of each of the independent variables: age, gender, or FIQ, or combinations of these variables, should be preserved to maximize the classification scores. A simplified form of the ComBat model is given by

$$\boldsymbol{Y} = \boldsymbol{\mu}_Y + \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\gamma} + \boldsymbol{\delta}\boldsymbol{\epsilon} \tag{5.9}$$

where $\boldsymbol{\mu}_Y$ is the mean value vector of $\boldsymbol{Y}$, and the vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ are parameters representing the additive and multiplicative site effects respectively [JLR07], the rest of the variables are equal to those defined by Equation 5.2. The vector of site adjusted values, $\hat{\boldsymbol{y}}$, is

$$\hat{\boldsymbol{y}} = \frac{\boldsymbol{y} - \hat{\boldsymbol{\mu}}_y - \boldsymbol{X}\hat{\boldsymbol{\beta}} + \boldsymbol{\gamma}^*}{\boldsymbol{\delta}^*} + \hat{\boldsymbol{\mu}}_y + \boldsymbol{X}\hat{\boldsymbol{\beta}} \tag{5.10}$$

where $\hat{\boldsymbol{\mu}}_y$, $\hat{\boldsymbol{\beta}}$, $\boldsymbol{\gamma}^*$ and $\boldsymbol{\delta}^*$ are estimated values of the corresponding parameters. The ComBat model removes the confounding effects introduced by site effects, and preserves the variability introduced by the independent variables included in the the design matrix $\boldsymbol{X}$ [FPT$^+$17].

We computed two sets of new features using the ComBat harmonization models given by Equations 5.9 and 5.10. The first set was obtained using functional connectivity as the dependent variable, and the second set using the Fisher z-transformation of the functional connectivity, $\boldsymbol{FC}_{FZ}$ (see Section 5.2.7), as the dependent variable. The new features are given by Equations 5.11 and 5.12, respectively.

$$\boldsymbol{cbX} = \frac{\boldsymbol{FC} - \hat{\boldsymbol{\mu}}_{FC} - \boldsymbol{X}\hat{\boldsymbol{\beta}} + \boldsymbol{\gamma}^*}{\boldsymbol{\delta}^*} + \hat{\boldsymbol{\mu}}_{FC} + \boldsymbol{X}\hat{\boldsymbol{\beta}} \tag{5.11}$$

$$\boldsymbol{cbX}_{FZ} = \frac{\boldsymbol{FC}_{FZ} - \hat{\boldsymbol{\mu}}_{FC_{FZ}} - \boldsymbol{X}\hat{\boldsymbol{\beta}} + \boldsymbol{\gamma}^*}{\boldsymbol{\delta}^*} + \hat{\boldsymbol{\mu}}_{FC_{FZ}} + \boldsymbol{X}\hat{\boldsymbol{\beta}} \tag{5.12}$$

The complete set of new features computed with the ComBat harmonization models and the corresponding independent variables are given in Table 5.2.

Algorithm 5.3 shows the implementation of the ComBat models using the functional connectivity values to compute the new feature ***cbAFG***.

Algorithm 5.3 Algorithm to implement the ComBat models using the functional connectivity values as dependent variables, and age, FIQ, and gender as independent variables, to compute the new feature ***cbAFG***, for the machine learning models (Section 5.2.3). The variables and functions are from the NeuroCombat models given in https://github.com/Jfortin1/neuroCombat.

```
1: Input: $\mathbf{FC_{ld}}$                                  ▷ $\mathbf{FC_{ld}}$ : FC list of dictionaries
2: Output: $\boldsymbol{cbAFG}$                               ▷ $\boldsymbol{cbAFG}$ : New Feature (ComBat output)
3: procedure CB($\mathbf{FC_{ld}}$)
4:     $\mathbf{covars} \leftarrow \mathbf{FC_{ld}}(\mathbf{age}, \mathbf{FIQ}, \mathbf{gender}, \mathbf{scanner})$
5:     $\mathbf{batch_{col}} \leftarrow \mathbf{FC_{ld}}(\mathbf{scanner})$
6:     $\mathbf{data} \leftarrow \mathbf{FC_{ld}}(\boldsymbol{FC})$
7:     $\mathbf{combat} \leftarrow \mathbf{NeuroCombat}(\mathbf{data}, \mathbf{covars}, \mathbf{batch_{col}})$
8:     $\boldsymbol{cbAFG} \leftarrow \mathbf{combat.data}$
9:     return $\boldsymbol{cbAFG}$
10: end procedure
```

5.2.7 Normalization methods

Considering that the independent variables of the multiple linear regression models and the ComBat harmonization models give only a partial explanation of the variability of the dependent variables, we also generated new features by implementing normalization methods through the transformation of the functional connectivity values in more statistically uniform new values, by reducing biases and outliers introduced by unknown confound variables [SS20].

For the mathematical definition of the normalization methods implemented in this study, we represented the functional connectivity, for the 1035 subjects of the 17 ABIDE sites (see Table 5.1), as a matrix with I = 1035 subjects as rows, and J = 19990 features as columns.

The normalization methods presented in this section are the Fisher z-transformation, as well as methods to compute new features by demeaning the functional connectivity values. All these methods were implemented with the goal of maximizing the classification scores by controlling the confounding effects of unknown variables related to all the sites.

Fisher z-transformation

The Fisher z-transformation was proposed by Fisher [Fis15] to correct for skewness (lack of symmetry) of the Pearson correlation coefficients, resulting in coefficients approximately normally distributed. We implemented this method because in this study, the functional connectivity values were computed as Pearson correlation coefficients, and any skewness of these values may be different between the data of the ABIDE sites with some potential confounding effects. The Fisher z-transformation is given by

$$\boldsymbol{FC}_{FZ} = \mathbf{arctanh}(\boldsymbol{FC}) \tag{5.13}$$

The new features obtained with the Fisher z-transformation of the functional connectivity, $\boldsymbol{FC}_{FZ}$, were computed as described in Sections 5.2.5 and 5.2.6, and summarized in Table 5.2.

Demeaning the functional connectivity(FC) values

We implemented normalization methods by demeaning the functional connectivity values with three different average values, resulting in three new corresponding normalization features: $\Delta\boldsymbol{avg}, \Delta\boldsymbol{avgSite}$, and $\Delta\boldsymbol{avgSubj}$.

The new features $\Delta\boldsymbol{avg}$ are given by

$$\Delta\boldsymbol{avg} = \boldsymbol{FC} - \boldsymbol{\mu}_{FC} \tag{5.14}$$

where the component $\mu_{FC,j} = \sum_{i=1}^{I} FC_{ij}/I$ of the vector $\boldsymbol{\mu}_{FC}$, is the average of the j^{th} component of the functional connectivity computed over all subjects of the 17 ABIDE sites. Algorithm 5.4 shows the computation of these new features.

The new features $\Delta\boldsymbol{avgSite}$ are given by

$$\Delta\boldsymbol{avgSite} = (\Delta\boldsymbol{avg}_{si_1}, \Delta\boldsymbol{avg}_{si_1}, ..., \Delta\boldsymbol{avg}_{si_{17}}) \tag{5.15}$$

where $\Delta\boldsymbol{avg}_{si_k} = \boldsymbol{FC}_{si_k} - \mu_{si_k}$ is the new vector of features, $\boldsymbol{FC}_{si_k}$ is the functional connectivity vector, and $\mu_{si_k}, k \leq 17$, is the average of all the values of functional connectivity, for the k^{th} site. Algorithm 5.5 shows the computation of these new features.

Algorithm 5.4 Algorithm to compute the new features $\Delta\boldsymbol{avg}$ by demeaning the functional connectivity values with the average of the functional connectivity computed over all the ABIDE subjects using the NumPy library(https://numpy.org/doc).

```
1: Input: $\boldsymbol{FC}$                                  ▷ $\boldsymbol{FC}$ : FC values for all the subjects
2: Input: Subjects                               ▷ Subjects : list of ABIDE subjects
3: Output: $\Delta\boldsymbol{avg}$                                   ▷ $\Delta\boldsymbol{avg}$ : New features
4: procedure FCAvg(Subjects, $\boldsymbol{FC}$)
5:     $\boldsymbol{fc}_{avg} \leftarrow []$                                   ▷ $\boldsymbol{fc}_{avg}$: empty list
6:     $\Delta\boldsymbol{FC}_{avg} \leftarrow []$                  ▷ $\Delta\boldsymbol{FC}_{avg}$: empty list for new features
7:     m $\leftarrow$ 19900              ▷ m: number of features for cc200 brain atlas
8:     for $k \rightarrow 1, m$ do
9:         $\boldsymbol{fc} \leftarrow []$                                      ▷ fc: empty list
10:        for $SUB \in Subjects$ do
11:            $\boldsymbol{fc}$.append $\leftarrow \boldsymbol{FC}(SUB)(k)$
12:        end for
13:        $\boldsymbol{fc}_{avg}$.append $\leftarrow$ numpy.mean($\boldsymbol{fc}$)
14:    end for
15:    for $SUB \in Subjects$ do
16:        $\Delta\boldsymbol{avg}[SUB] \leftarrow \boldsymbol{FC}(SUB) - \boldsymbol{fc}_{avg}$
17:    end for
18:    return $\Delta\boldsymbol{avg}$
19: end procedure
```

The new features $\Delta\boldsymbol{avgSubj}$ are given by

$$\Delta\boldsymbol{avgSubj} = \boldsymbol{FC} - \boldsymbol{\mu}_{FC_{Sub}} \tag{5.16}$$

where the component $\mu_{FC_{Subj,i}} = \sum_{j=1}^{J} FC_{i,j}/J$ of the vector $\boldsymbol{\mu}_{FC_{Subj}}$, is the average of the functional connectivity values computed for the i^{th} subject. Algorithm 5.6 shows the computation of these new features.

Algorithm 5.5 Algorithm to compute the new features Δ***avgSite*** by demeaning the functional connectivity values with the average computed over all the values of functional connectivity of an ABIDE site using the NumPy library (https://numpy.org/doc).

```
 1: Input: $\boldsymbol{FC}$                                        ▷ $\boldsymbol{FC}$ : FC values for all the subjects
 2: Input: Sites                                          ▷ Sites : list of ABIDE sites
 3: Input: Subjects                                  ▷ Subjects : list of ABIDE subjects
 4: Output: $\Delta$avgSite                                     ▷ $\Delta$avgSite : New features
 5: procedure FCAVGSITE(Sites, Subjects, $\boldsymbol{FC}$ )
 6:     for $SITE \in Sites$ do
 7:         $\boldsymbol{fc} \leftarrow []$                                            ▷ $\boldsymbol{fc}$: empty list
 8:         $\boldsymbol{fc}_{avg} \leftarrow []$                                       ▷ $\boldsymbol{fc}_{avg}$: empty list
 9:         for $SUB \in SITE(Subjects)$ do
10:             $\boldsymbol{fc}$.append $\leftarrow$ FC($SUB$)
11:         end for
12:         $\boldsymbol{fc}_{avg}$.append $\leftarrow$ numpy.mean($\boldsymbol{fc}$)
13:         for $SUB \in SITE(Subjects)$ do
14:             $\Delta$avgSite($SUB$) $\leftarrow \boldsymbol{FC}(SUB) - \boldsymbol{fc}_{avg}$
15:         end for
16:     end for
17:     return $\Delta$avgSite
18: end procedure
```

5.2.8 Sub-samples selection

A common practice in machine learning analysis is to compare computed classification accuracies with those obtained by chance level, i.e., by assuming the uniform distribution that a subject may be classified as patient or control. For this binary classification problem, the chance level is equal to 50 %, if the sample has infinite size. Reference [CJ15] showed that for small data sets (less than 200 samples), the empirical chance level computed from random classification was greater than the theoretical chance level for an infinite sample, for example, for a sample size of 100, the chance level accuracy was 58.0 % at a significance level of $p < 0.05$, and for a sample size of 60 was 60 % at a significance level of $p < 0.05$. Considering these limits, the sizes of a high percentage of the selected sub-samples presented in this paper were greater than 100 subjects, and when the sub-samples contained less than 100 subjects, the corresponding accuracies were much greater than 58 % (see Table 5.6).

Algorithm 5.6 Algorithm to compute the new features $\Delta \boldsymbol{avgSubj}$ by demeaning the functional connectivity values with the average computed for each ABIDE subject using the Numpy library (https://numpy.org/doc).

```
1:  Input: $\boldsymbol{FC}$                                   ▷ $\boldsymbol{FC}$ : FC values for all the subjects
2:  Input: Subjects                                  ▷ Subjects : list of ABIDE subjects
3:  Output: $\Delta \boldsymbol{avgSubj}$                             ▷ $\Delta \boldsymbol{avgSubj}$ : New features
4:  procedure FCavgSub(Subjects, $\boldsymbol{FC}$ )
5:      $\boldsymbol{fc} \leftarrow []$                                        ▷ $\boldsymbol{fc}$: empty list
6:      $\boldsymbol{fc}_{avg} \leftarrow []$                                  ▷ $\boldsymbol{fc}_{avg}$: empty list
7:      for $SUB \in Subjects$ do
8:          $\boldsymbol{fc}$.append $\leftarrow$ $\mathbf{FC}(SUB)$
9:      end for
10:     $\boldsymbol{fc}_{avg}$.append $\leftarrow$ numpy.mean($\boldsymbol{fc}$)
11:     for $SUB \in Subjects$ do
12:         $\Delta \boldsymbol{avgSubj}(SUB) \leftarrow \boldsymbol{FC}(SUB) - \boldsymbol{fc}_{avg}(SUB)$
13:     end for
14:     return $\Delta \boldsymbol{avgSubj}$
15: end procedure
```

The stratification methods used to generate the baseline and the homogeneous sub-samples implemented in this study were based on the stratification techniques defined by [Par14] and [Ney92], which basically consists of probability sampling methods on which the subjects of the target population are divided into sub-samples or strata where within each sub-sample the subjects have similar characteristics. The criteria used to select the sites or subjects included in these sub-samples were suitable to accomplish the goal of maximizing the classification scores computed with the machine learning analysis of the rs-fMRI multi-site data. These criteria were defined in a different and simplified way than those established in the works of [Par14] and [Ney92].

Homogeneous sub-samples selection

In this study, we selected homogeneous sub-samples integrated with subjects classified by ranges of age and ranges of full IQ (FIQ). The first eight homogeneous sub-samples given in Table 5.3 were formed by grouping subjects with the same range of ages, or of FIQ, the last two sub-samples were formed with the intersection of subjects with selected ranges of these phenotypic values.

Algorithm 5.7 shows an example of the computation of the total number of subjects, the number of patients, and the number of control subjects in a homogeneous sub-sample with ABIDE subjects with ages between 10 and 20 years.

Table 5.3: Homogeneous sub-samples formed by grouping subjects with the same range of ages, FIQ, or gender as described in Section 5.2.8.

Sub-sample	**Acronym**	**C/A/T**
$0 < \textbf{age} < 10$	**age-10**	74/69/143
$10 < \textbf{age} \leq 15$	**age-1015**	209/203/412
$15 < \textbf{age} \leq 20$	**age-1520**	115/110/225
$10 < \textbf{age} \leq 20$	**age-1020**	324/313/637
$20 < \textbf{age}$	**age-20**	132/123/255
$0 < \textbf{FIQ} \leq 89$	**FIQ-89**	24/92/116
$89 < \textbf{FIQ} \leq 110$	**FIQ-89110**	238/215/453
$110 < \textbf{FIQ}$	**FIQ-110**	268/198/466
$(10 < \textbf{age} \leq 20) \cap (0 < \textbf{FIQ} \leq 89)$	**age-1020-FIQ-89**	21/67/88
$(10 < \textbf{age} \leq 20) \cap (89 < \textbf{FIQ} \leq 110)$	**age-1020-FIQ-89110**	153/138/291

Algorithm 5.7 Algorithm to compute the total number of subjects, the number of patients, and the number of control subjects in a homogeneous sub-sample with ABIDE subjects with ages between 10 and 20 years.

```
1: Input: Sites                                   ▷ Sites : list of ABIDE sites
2: Input: Subjects                                ▷ Subjects : list of ABIDE subjects
3: Output: $age_s, age_{sc}, age_{sa}$            ▷ counters with the number of subjects/site in the
   sub-sample
4: Output: $age, age_c, age_a$                    ▷ counters with the total number of subjects in the
   sub-sample
5: procedure AGE(Sites, Subjects)
6:     for $SITE \in Sites$ do
7:         for $SUB \in SITE(Subjects)$ do
8:             if $SUB$ is autistic then
9:                 if $10 < SUB(age) \leq 20$ then
10:                    $age_s \leftarrow age + 1$
11:                    $age_{sa} \leftarrow age_a + 1$
12:                end if
13:            else
14:                if $10 < SUB(age) \leq 20$ then
15:                    $age_s \leftarrow age + 1$
16:                    $age_{sc} \leftarrow age_c + 1$
17:                end if
18:            end if
19:        end for
20:        $age \leftarrow age + age_s$
21:        $age_a \leftarrow age_a + age_{sa}$
22:        $age_c \leftarrow age_c + age_{sc}$
23:        return $age_s, age_{sc}, age_{sa}$
24:    end for
25:    return $age, age_c, age_a$
26: end procedure
```

Baseline sub-samples selection

We formed a baseline set of sub-samples by progressively selecting the sites with the greatest values of accuracy computed with ASD-DiagNet, i.e., the sub-sample with 4 sites was integrated by the first four sites of Table 5.4. The baseline classification scores computed with ASD-DiagNet, using the functional connectivity values of the subjects grouped in these sub-samples, are given in Table 5.5, on which the number of control (C), autistic (A) and total (T) subjects are included to compare the sizes

Table 5.4: Values and standard deviations of the classification scores computed with ASD-DiagNet (see Section 5.2.3) for each ABIDE site, where the functional connectivity values were used as features, and cc200 as the brain atlas. The classification scores computed with the whole 17 ABIDE sites are included for comparison.

Site	**Accuracy**	**Sensitivity**	**Specificity**
Olin	81.2 ± 2.7	90.5 ± 2.7	70.0 ± 4.5
OHSU	76.8 ± 2.4	92.7 ± 2.0	63.0± 4.6
whole 17 sites	70.2 ± 0.1	68.8 ± 0.6	71.6 ± 0.2
KKI	70.1 ± 1.7	29.5 ± 1.5	98.7 ± 2.2
USM	70.0 ± 1.4	92.4 ± 2.2	28.8 ± 3.0
NYU	66.8 ± 1.1	51.6 ± 2.1	78.2 ± 1.7
UCLA	66.4 ± 0.9	72.9 ± 1.2	58.8 ± 1.7
Yale	64.6 ± 2.1	58.7 ± 1.6	70.2 ± 4.3
Stanford	63.9 ± 3.4	47.3 ± 3.7	81.0 ± 4.9
CMU	63.8 ± 4.7	60.7 ± 10.0	66.0 ± 5.3
UM	63.4 ± 0.6	48.9 ± 1.2	76.5 ± 0.9
Leuven	62.4 ± 2.7	55.2 ± 3.5	68.7 ± 3.4
Pitt	61.4 ± 2.4	67.0 ± 3.7	55.4 ± 2.7
SDSU	55.9 ± 1.7	15.3 ± 3.1	82.6 ± 1.2
SBL	55.0 ± 3.7	54.7 ± 4.0	55.3 ± 5.2
MaxMun	54.0 ± 1.5	24.2 ± 1.8	81.8 ± 1.7
Caltech	52.1 ± 2.1	58.7 ± 2.3	48.5 ± 3.7
Trinity	44.6 ± 1.8	21.6 ± 2.7	65.2 ± 2.6

of the sub-samples. The last row of Table 5.5 shows the classification scores obtained with the machine learning models presented in [HFC$^+$18] for 17 ABIDE sites. These results showed the existence of confounding effects affecting the classification scores between sites. Furthermore, the baseline classification scores computed with the sub-samples were always greater than the scores computed with the whole 17 sites.

The baseline sub-samples and the corresponding baseline classification scores provided a convenient framework by comparing the classification scores obtained with the new features defined in Sections 5.2.5, 5.2.6 and 5.2.7.

Table 5.5: Values and standard deviations of the baseline classification scores (accuracy (Ac), sensitivity (Se), and specificity (Sp)) computed with ASD-DiagNet.

Sub-sample	**C/A/T**	**Ac**	**Se**	**Sp**
10-sites	361/353/714	**73.5 ± 0.6**	**71.6 ± 0.5**	**75.5 ± 0.8**
8-sites	274/273/547	**73.2 ± 0.7**	**73.3 ± 1.1**	**73.2 ± 0.5**
9-sites	287/287/574	**72.8 ± 0.4**	**72.2 ± 0.9**	**73.5 ± 0.2**
4-sites	82/97/179	**72.8 ± 0.3**	**76.5 ± 0.7**	**68.5 ± 0.3**
7-sites	254/254/508	**72.6 ± 0.4**	**73.2 ± 0.8**	**71.9 ± 0.3**
13-sites	444/425/869	**72.4 ± 0.3**	**69.7 ± 0.5**	**74.9 ± 0.3**
6-sites	226/226/452	**72.1 ± 0.7**	**71.5 ± 0.5**	**72.6 ± 1.0**
11-sites	395/382/777	**71.7 ± 0.1**	**68.9 ± 0.3**	**74.5 ± 0.3**
14-sites	459/440/899	**71.5 ± 0.2**	**70.1 ± 0.5**	**72.8 ± 0.7**
15-sites	487/464/951	**71.4 ± 0.1**	**69.2 ± 0.2**	**73.5 ± 0.3**
5-sites	182/172/354	**71.2 ± 0.9**	**70.1 ± 0.5**	**72.1 ± 1.3**
12-sites	422/411/833	**71.4 ± 0.2**	**68.7 ± 0.3**	**74.1 ± 0.4**
16-sites	505/483/988	**70.8 ± 0.3**	**69.1 ± 0.7**	**72.4 ± 0.2**
whole 17 sites	530/505/1035	**70.2 ± 0.1**	**68.8 ± 0.6**	**71.6 ± 0.6**
Heinsfeld et.al.	530/505/1035	**70**	**74**	**63**

5.2.9 Methods for the statistical comparison of experimental results computed with the new features

Considering the strong dependence of the classification scores on the new features used to compute them, we performed statistical tests and computed the Wasserstein distance to compare the baseline classification scores, with those scores computed with the new features obtained with the models and the normalization methods described in Sections 5.2.5, 5.2.6, and 5.2.7, respectively. All these classification scores were computed with the ASD-DiagNet machine learning classifier (see Section 5.2.3).

To ensure the consistency of the statistical results, three statistical test methods were implemented to perform this statistical analysis. The chosen methods were: The parametric t-test (tt), and two nonparametric tests: the Kolmogorov–Smirnov test (kst), and the Mann–Whitney U test (mwt). The t-test was used to determine if the means of two sets of data were statistically different from each other. The nonparametric tests computed several test statistics to determine if two sets of data are samples of the same distribution. All the statistics methods were implemented in the stats sub-package of the SciPy library in Python (https://scipy.org), more details about these methods in [TD00, CF14, SS16].

The main limitation of the statistical tests described above was that the comparison of the classification scores ignored the strong dependence of these scores on the sub-samples. Hence, to rank the new features according to the corresponding values of the classification scores for each sub-sample, we computed the percentage difference, for each sub-sample, between the classification scores ($\mathbf{CS}_{nf}$) computed with the new features and the baseline classification scores ($\mathbf{CS}_{bl}$), namely:

$$\mathbf{\Delta} = (\mathbf{CS}_{nf} - \mathbf{CS}_{bl})/\mathbf{CS}_{bl} * 100 \tag{5.17}$$

The following positive and negative ranges for these differences were defined: $0 < \mathbf{\Delta} \leq 2.0$(p1), $2.0 < \mathbf{\Delta} \leq 3.0$(p2), $3.0 < \mathbf{\Delta} \leq 4.0$(p3), $4.0 < \mathbf{\Delta}$(p4), $0 > \mathbf{\Delta} \geq -2.0$(n1), $-2.0 > \mathbf{\Delta} \geq -3.0$(n2), $-3.0 > \mathbf{\Delta} \geq -4.0$(n3), and $-4.0 > \mathbf{\Delta}$(n4). We then binned the number of values falling in each range in positive and negative bins. The values in these bins allowed us to rank the classification scores obtained with the new features, with the maximum rank assigned to those with the greatest number of values in the positive bins.

5.3 Results

We performed a comprehensive set of experiments to compute the classification scores with the new features obtained with the models and methods described in Sections 5.2.5, 5.2.6 and 5.2.7, using ASD-DiagNet as the machine learning classifier. For these experiments, we used a total of nineteen new features, as well as the fourteen baseline sub-samples given in Table 5.5, to obtain a total of 266 independent experimental results. We compared these results using the statistical methods described in Section 5.2.9. We also selected the sub-sample with which we obtained the maximum value of the classification scores obtained with each feature. We also presented the classification scores computed for the ten homogeneous sub-samples given in Table 5.3. To compare the experimental results with a different machine learning model, we computed classification scores with ASD-SAENet (see Section 5.2.3) using a selected set of the new features. Since, as far as we know, it is the first time our proposed baseline and homogeneous sub-samples have been implemented and used in this type of study, there are no similar published results to compare our experimental results. A detailed analysis of all the computed results is given in the following sections.

All the experiments presented in this work were performed on a Linux server with Ubuntu operating system version 16.04.6, 22 Intel Xeon Gold 6152 processors, a clock speed 2.1 GHz, and 125 GB of RAM. The GPU in this server was an NVIDIA Titan Xp, with 30 SM, 128 cores/SM, maximum clock rate of 1.58 GHz, 12196 MB of global memory, and CUDA version 11.4 with CUDA capability of 6.1.

5.3.1 Experimental results: Homogeneous sub-samples

Table 5.6 shows the values and standard deviations of the classification scores, computed with ASD-DiagNet for each of the homogeneous sub-samples of the 17 ABIDE sites given in Table 5.3. These values were computed using the values of functional connectivity as features, and the cc200 as the brain atlas. Only the sub-samples for which the accuracy was equal to or greater than 70 % are included. In general, the accuracy and sensitivity scores obtained with these sub-samples were greater than the baseline scores computed with the whole 17 ABIDE sites. The first two

Table 5.6: Values and standard deviations of the classification scores, computed with ASD-DiagNet for each of the homogeneous sub-samples of the 17 ABIDE sites given in Table 5.3 and described in Section 5.3.1. The baseline classification scores computed for the whole 17 ABIDE sites are included for comparison. The number of control (C), autistic (A), and total (T) subjects are included to compare the sizes of the sub-samples.

Sub-sample	**C/A/T**	**Accuracy**	**Sensitivity**	**Specificity**
FIQ-89	24/92/106	**85.9 ± 0.2**	**98.9 ± 0.1**	**34.2± 1.6**
age-1020-FIQ-89	21/67/88	**84.6 ± 0.3**	**99.6 ± 0.4**	**36.8 ± 2.7**
age-1020-FIQ-89-bal	65/67/132	**76.4 ± 0.7**	**82.3 ± 0.7**	**68.5 ± 0.8**
FIQ-89-bal	58/92/150	**76.0 ± 0.4**	**82.9 ± 0.3**	**65.1 ± 0.7**
age-1520	115/110/225	**72.0 ± 0.2**	**70.9 ± 0.5**	**73.1 ± 0.8**
age-1020	324/313/637	**71.9 ± 0.1**	**71.4 ± 0.4**	**72.4 ± 0.2**
FIQ-89-110	238/215/453	**70.3 ± 0.5**	**64.7 ± 0.8**	**75.4 ± 0.4**
whole 17 sites	530/505/1035	**70.2 ± 0.1**	**68.8 ± 0.6**	**71.6 ± 0.6**

sub-samples of Table 5.6, which include subjects with $0 <$ **FIQ** ≤ 89 obtained the maximum values of accuracy (85.9 %) and sensitivity (99.6 %), but they were unbalanced in the number of autistic and control subjects, inducing overfitting of the machine learning model and unbalanced sensitivity and specificity scores. We performed experiments to correct these unbalances by increasing the number of control subjects, randomly selected out of the **FIQ-89** and **age-10-20-FIQ-89** sub-samples. The classification scores computed with 34 and 44 additional control subjects in the sub-samples **FIQ-89-bal**, **age-10-20-FIQ-89-bal** sub-samples in Table 5.6, respectively, showed how these classification scores were lower but more balanced than those obtained with the original sub-samples. These sub-samples also obtained the maximum values of accuracy (76.4 %, 8.8 % above the baseline accuracy) and sensitivity (82.9 %, 20.5 % above the baseline sensitivity) among all the classification scores presented in this study.

5.3.2 Statistical comparison of experimental results computed with the new features

Table 5.7 shows the P-values obtained from statistical tests and the Wasserstein distance (**wa-d**) to compare the baseline classification scores, with those scores computed with the new features as defined in Section 5.2.9. Only the new features for which at least two P-values were less than 0.05 were included.

To rank the new features accordingly to the corresponding values of the classification scores for each sub-sample (see Section 5.2.9), the total values in the positive and negative bins obtained for the accuracy, sensitivity, and specificity scores, computed for each new feature, are summarized in Figure 5.3, which provides an efficient visualization of the rank of the classification scores obtained with the new features relative to the baseline classification scores.

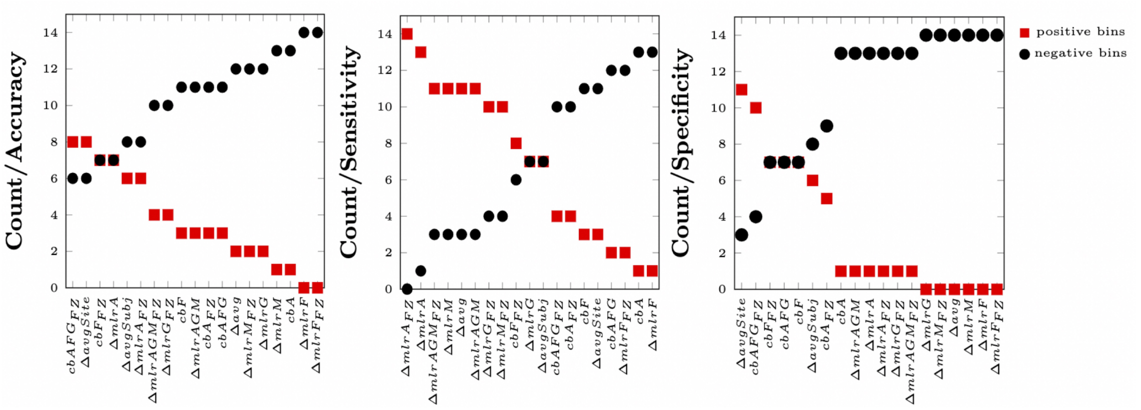


Figure 5.3: Summary of total counts of the number of values in the positive and negative bins in the ranges defined in Section 5.2.9, corresponding to the classification scores computed with ASD-DiagNet with the new features.

5.3.3 Experimental results: New features

We implemented a total of nineteen new features, ten of them using the multiple linear regression models defined in Section 5.2.5 and six using the ComBat harmonization models described in Section 5.2.6 (See Table 5.2). We also implemented three new features with the normalization methods described in Section 5.2.7. These new features were used to perform experiments to compute the classification scores with ASD-DiagNet for each of the baseline sub-samples described in Section 5.2.8, for which the baseline classification scores, obtained from the functional connectivity values, are given in Table 5.5. Table 5.8 summarizes the maximum values of these classification scores obtained with each new feature and with the corresponding baseline sub-sample.

Experimental results: Multiple linear regression models

The classification scores computed with the new features obtained with the multiple linear regression models (Section 5.2.5) on which each one of the individual independent variables age, FIQ, gender or MRI vendor were regressed out to obtain the new MLR features of Table 5.2, are given in Figures 5.4 and 5.5, on which they are compared to the baseline classification scores given in Table 5.5.

Three of the maximum accuracy scores and four of the maximum sensitivity scores (see Table 5.8) were obtained with the new features computed with the multiple linear regression models. Seven of these features were among the first eight features with the maximum counts in the positive bins for sensitivity (see Figure 5.3).

Table 5.7: P-values obtained from statistical tests and the Wasserstein distance (**wa-d**) defined in Section 5.2.9. All the classification scores were computed with ASD-DiagNet for the sub-samples of Table 5.5. Only the features for which at least two P-values were less than 0.05 were included.

Feature	**Score**	**kst**	**tt**	**mwt**	**Wa-d**
$\boldsymbol{\Delta mlrA}$	**Accuracy**	0.92	0.78	0.73	0.002
	Sensitivity	0.15	0.14	0.18	0.014
	Specificity	0.15	0.04	0.04	0.018
$\boldsymbol{\Delta mlrA}_{FZ}$	**Accuracy**	0.34	0.07	0.12	0.003
	Sensitivity	0.15	0.06	0.05	0.017
	Specificity	0.15	0.03	0.03	0.019
$\boldsymbol{\Delta mlrF}$	**Accuracy**	10^{-7}	10^{-10}	10^{-5}	0.041
	Sensitivity	0.06	0.02	0.01	0.02
	Specificity	10^{-6}	10^{-8}	10^{-5}	0.061
$\boldsymbol{\Delta mlrF}_{FZ}$	**Accuracy**	10^{-7}	10^{-10}	10^{-5}	0.045
	Sensitivity	0.02	0.01	0.01	0.024
	Specificity	10^{-6}	10^{-9}	10^{-5}	0.066
$\boldsymbol{\Delta mlrG}$	**Accuracy**	0.15	0.05	0.1	0.011
	Sensitivity	0.06	0.91	0.54	0.014
	Specificity	0.001	0.001	0.001	0.024
$\boldsymbol{\Delta mlrG}_{FZ}$	**Accuracy**	0.34	0.14	0.21	0.007
	Sensitivity	0.92	0.53	0.45	0.006
	Specificity	0.06	0.01	0.01	0.02
$\boldsymbol{\Delta mlrM}$	**Accuracy**	0.34	0.04	0.06	0.009
	Sensitivity	0.64	0.28	0.26	0.012
	Specificity	0.001	0.002	0.0004	0.029
$\boldsymbol{\Delta mlrM}_{FZ}$	**Accuracy**	0.34	0.04	0.06	0.009
	Sensitivity	0.34	0.37	0.37	0.009
	Specificity	0.001	0.003	0.001	0.027
$\boldsymbol{\Delta mlrAGM}$	**Accuracy**	0.15	0.24	0.16	0.006
	Sensitivity	0.64	0.33	0.26	0.011
	Specificity	0.005	0.005	0.002	0.019
$\boldsymbol{\Delta mlrAGM}_{FZ}$	**Accuracy**	0.15	0.24	0.18	0.007
	Sensitivity	0.64	0.3	0.28	0.01
	Specificity	0.02	0.01	0.002	0.021
$\boldsymbol{cbA}$	**Accuracy**	0.15	0.01	0.02	0.013
	Sensitivity	0.34	0.07	0.09	0.016
	Specificity	0.34	0.18	0.19	0.01
$\boldsymbol{cbA}_{FZ}$	**Accuracy**	0.06	0.02	0.02	0.011
	Sensitivity	0.34	0.05	0.14	0.015
	Specificity	0.64	0.45	0.40	0.007
$\boldsymbol{\Delta avg}$	**Accuracy**	0.34	0.11	0.14	0.008
	Sensitivity	0.34	0.43	0.30	0.01
	Specificity	0.005	0.003	0.001	0.023

Table 5.8: The maximum values of the classification scores (accuracy (Ac), sensitivity (Se) and specificity (Sp)) computed with ASD-DiagNet using the new features obtained with the MLR models, ComBat models, and normalization methods, and the corresponding baseline sub-samples (SS) (see Table 5.5). The percentage difference between the results obtained with the new features and the baseline classification scores obtained for the whole 17 sites are included. The five greatest values for each classification score are highlighted in bold.

Feature	**Ac**(SS)	%	**Se**(SS)	%	**Sp**(SS)	%
$\boldsymbol{\Delta mlrAGM}$	**74.3**± 0.2(7)	**5.8**	75.2 ± 0.3(7)	9.3	73.5 ± 0.3(7)	2.7
$\boldsymbol{\Delta mlrAGM}_{FZ}$	**74.2** ± 0.7(8)	**5.7**	76.4 ± 0.5(4)	11.1	72.7 ± 0.7(8)	1.5
$\boldsymbol{\Delta mlrA}_{FZ}$	**74.1** ± 0.3(10)	**5.6**	**78.1** ± 0.5(4)	**13.5**	73.8 ± 0.2(10)	3.1
$\boldsymbol{cbAFG}_{FZ}$	**74.1** ± 0.1(10)	**5.6**	74.4± 0.5(4)	8.1	**76.7** ± 0.4(12)	**7.1**
$\boldsymbol{\Delta avgSite}$	**73.8** ± 0.2(10)	**5.1**	77.1 ± 0.6(4)	12.1	**77.0** ± 0.2(10)	**7.5**
$\boldsymbol{\Delta mlrA}$	73.6 ± 0.2(8)	4.8	77.1 ± 1.0(4)	12.1	73.4 ± 0.2(10)	2.5
$\boldsymbol{\Delta avg}$	73.5 ± 0.3(9)	4.8	**77.4** ± 0.2(4)	**12.5**	73.4 ± 0.4(9)	2.5
$\boldsymbol{\Delta mlrG}_{FZ}$	73.1 ± 0.2(10)	4.1	**78.1** ± 0.5(4)	**13.5**	73.4 ± 0.6(12)	2.5
$\boldsymbol{cbF}$	73.0 ± 0.2(10)	4.0	75.3 ± 0.6(4)	9.4	**76.0** ± 0.8(13)	**6.1**
$\boldsymbol{\Delta avgSubj}$	72.8 ± 0.1(9)	3.7	74.2 ± 0.4(4)	7.8	74.4 ± 1.2(14)	3.9
$\boldsymbol{\Delta mlrG}$	72.7 ± 0.2(9)	3.6	**78.6** ± 1.2(4)	**14.2**	72.8 ± 0.3(13)	1.7
$\boldsymbol{\Delta mlrM}$	72.7 ± 0.1(8)	3.6	**78.0** ± 0.4(4)	**13.4**	72.7± 0.4(11)	1.6
$\boldsymbol{\Delta mlrM}_{FZ}$	72.7 ± 0.1(9)	3.6	76.5 ± 0.7(4)	11.2	72.9 ± 0.4(13)	1.8
$\boldsymbol{cbAFG}$	72.7 ± 0.1(10)	3.6	75.6 ± 0.5(4)	9.9	**75.0** ± 0.1(15)	**4.8**
$\boldsymbol{cbA}_{FZ}$	72.7 ± 0.1(10)	3.6	71.9 ± 0.6(4)	4.5	74.2± 0.2(10)	3.6
$\boldsymbol{cbF}_{FZ}$	72.6 ± 0.1(10)	3.4	76.7 ± 0.6(4)	11.5	**75.6** ± 0.5(14)	**5.6**
$\boldsymbol{cbA}$	72.5 ± 0.5(10)	3.3	74.1 ± 1.1(4)	7.7	74.8 ± 0.5(13)	4.5
$\boldsymbol{\Delta mlrF}$	70.0 ± 0.4(10)	-0.3	73.8 ± 1.3(4)	7.3	69.9± 0.6(13)	-2.4
$\boldsymbol{\Delta mlrF}_{FZ}$	69.6 ± 0.3(10)	-0.9	73.5 ± 1.4(4)	6.8	69.5 ± 0.2(13)	-2.9
$\boldsymbol{FC(whole)}$	70.2		68.8		71.6	

Our experiments also showed that the specificity scores computed with the new features obtained with the multiple linear regression models, were below the baseline specificity scores for almost all the sub-samples, except sub-sample 7, as shown in Figures 5.4 and 5.5. More details about the results obtained with these features follow.

The first main result obtained with the multiple linear regression models was that all the classification scores computed with the new features $\boldsymbol{\Delta mlrF}$ and $\boldsymbol{\Delta mlrF}_{FZ}$ obtained when the FIQ variables were regressed out (see Table 5.2), were smaller

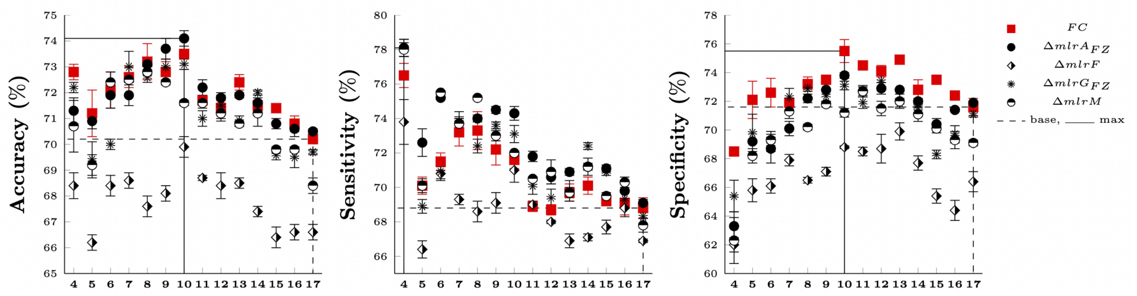


Figure 5.4: Classification scores computed with ASD-DiagNet, using selected new features obtained from the multiple linear regression models with individual independent variables described in Section 5.2.5, compared with the baseline classification scores (FC) given in Table 5.5. The baseline values for the whole 17 sites are indicated by the dashed line, while the maximum values are indicated by the continuous line.

than the baseline classification scores shown in Figure 5.4. This result was also confirmed by the p-values given in Table 5.7, and the counts in the negative bins summarized in Figure 5.3, obtained by the classification scores computed with these features.

The second main result was the quantification of the confounding effects of the variables age, gender, or MRI vendor. The results of the experiments showed that the new features on which age was regressed out, $\boldsymbol{\Delta mlrA}$ and $\boldsymbol{\Delta mlrA}_{FZ}$, were among the first six features with the maximum accuracy values given in Table 5.8. These features were also among the first six features and the first two features with the maximum counts in the positive bins for accuracy and sensitivity given in Figure 5.3, respectively. Figure 5.4 shows that the accuracy scores computed with the feature $\boldsymbol{\Delta mlrA}_{FZ}$ were greater than six of the baseline accuracy scores and that the sensitivity scores computed with this feature were greater than all the baseline sensitivity scores, with a maximum value of sensitivity, computed for the sub-sample 4, of 78.1 %, 13.5 % above the baseline value for the whole 17 sites (see Table 5.8).

The results of the experiments also showed that the new feature on which the gender variable was regressed out, $\Delta\boldsymbol{mlrG}_{FZ}$, was among the first eight features with the maximum accuracy values given in Table 5.8. This feature was also among the first seven features with the maximum counts in the positive bins for the sensitivity score given in Figure 5.3. Figure 5.4 shows that the sensitivity scores computed with this feature were greater than ten of the baseline sensitivity scores. Another important result was that the sensitivity score computed with the feature $\Delta\boldsymbol{mlrG}$ obtained a maximum value among all the sensitivity scores obtained with the new features, computed for the sub-sample 4, of 78.6 %, 14.2 % above the baseline value for the whole 17 sites (see Table 5.8).

Table 5.8 shows that the sensitivity score computed with the new feature on which the MRI vendor variable was regressed out, $\Delta\boldsymbol{mlrM}$, was among the first three maximum sensitivity values given in the table. This feature was also among the first seven features with the maximum counts in the positive bins for sensitivity given in Figure 5.3. Figure 5.4 shows that the sensitivity scores computed with this feature were greater than eleven of the baseline sensitivity scores, with a maximum sensitivity score for the sub-sample 4, of 78.0 %, 13.4 % above the baseline value for the whole 17 sites (see Table 5.8).

Additional and important results were computed with the new features $\Delta\boldsymbol{mlrAGM}$ and $\Delta\boldsymbol{mlrAGM}_{FZ}$ which were obtained with the multiple linear regression models with age, gender and MRI vendor as independent variables. The accuracy scores computed with these features were the maximum values of accuracy among all the features (see Table 5.8), with a maximum value of 74.3 % (5.8 % above the baseline value) for the sub-sample with 7 sites. Figure 5.5 shows that the sensitivity scores computed with these features were greater than eleven of the baseline sensitivity scores, with a maximum value of 76.4 % (11.1 % above the baseline value) shown in

Table 5.8. In general, all the results obtained with the new features computed with the multiple linear regression models were confirmed by the P-values given in Table 5.7.

Classification scores computed with ASD-SAENet: Figure 5.6 gives an example of the classification scores computed with ASD-SAENet using selected new features. The comparison of these results with those obtained with ASD-DiagNet using the same features (see Figure 5.5), showed that the classification scores obtained in these experiments were strongly dependent on the machine learning model used for these computations.

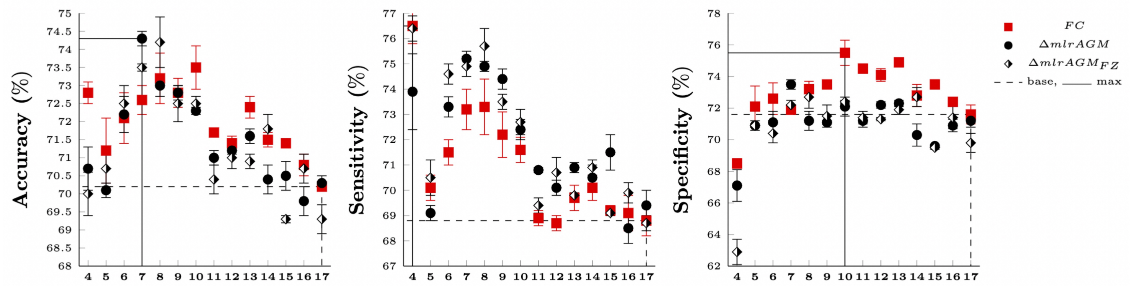


Figure 5.5: Classification scores computed with ASD-DiagNet using selected new features obtained from the multiple linear regression models described in Section 5.2.5, compared with the baseline classification scores given in Table 5.5. The baseline values for the whole 17 sites are indicated by the dashed line, while the maximum values are indicated by the continuous line.

Experimental results: ComBat harmonization models

The classification scores computed with the new features obtained with the ComBat harmonization models (Section 5.2.6) given in Table 5.2, are shown in Figure 5.7 on which they are compared to the baselines classification scores given in Table 5.5. One of the maximum accuracy scores and four of the maximum specificity scores (see Table 5.8) were obtained with the new features computed with the ComBat

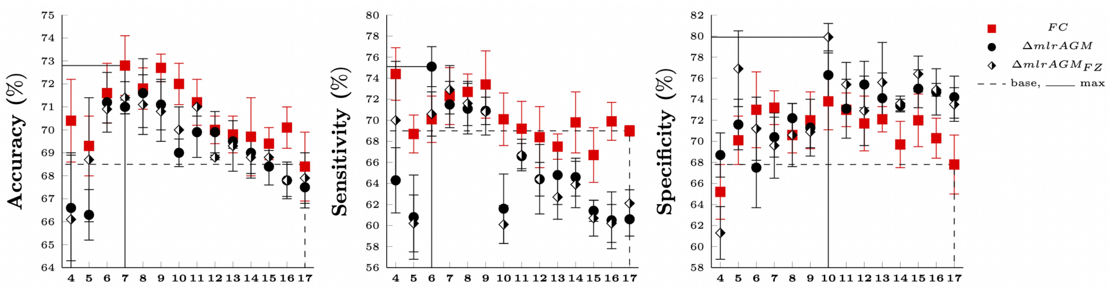


Figure 5.6: Classification scores computed using selected new features obtained from the multiple linear regression models described in Section 5.2.5, compared with the baseline classification scores computed with the functional connectivity values and the sub-samples given in Table 5.5. All the classification scores were computed with the ASD-SAENet machine learning model (see Section 5.2.3). The baseline values for the whole 17 sites are indicated by the dashed line, while the maximum values are indicated by the continuous line.

models. Two of these features were also among the first three features and four of them were among the first five features with the maximum counts in the positive bins for accuracy and specificity (see Figure 5.3), respectively. More details about the results obtained with these features follow. The new feature $\boldsymbol{cbAFG}_{FZ}$ obtained

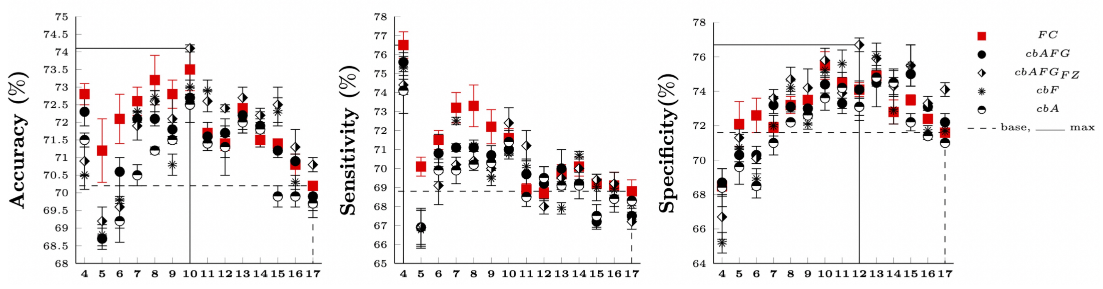


Figure 5.7: Classification scores computed with ASD-DiagNet using selected new features obtained from the ComBat harmonization models described in Section 5.2.6, compared with the baseline classification scores (FC) given in Table 5.5. The baseline values for the whole 17 sites are indicated by the dashed line, while the maximum values are indicated by the continuous line.

with the ComBat models (see Table 5.2) on which the variability introduced by the phenotypic variables age, FIQ, and gender was conserved, was among the first four features with maximum accuracy and maximum specificity values given in Table 5.8.

Figure 5.7 shows that the accuracy scores computed with this feature were greater than the baseline accuracy scores computed with sub-samples 10 to 16, as well as with the whole 17 sites. The specificity scores computed with this feature were greater than ten of the baseline specificity scores, obtaining the second maximum value of 76.7 % (7.1 % above the baseline value) shown in Table 5.8. This feature also obtained the maximum value of the counts in the positive bins for accuracy and the second maximum value for specificity given in Figure 5.3. The fifth maximum value of the specificity score, 75.0 % (4.8 % above the baseline value) given in Table 5.8 was computed with the new feature ***cbAFG***. This feature was also among the first four features with the maximum values of the counts in the positive bins for specificity given in Figure 5.3.

The new feature ***cbF*** obtained with the ComBat models (see Table 5.2) on which the variability introduced by the FIQ variable was conserved, was among the first four features with maximum specificity values given in Table 5.8. Figure 5.7 shows that the specificity scores computed with this feature were greater than seven of the baseline specificity scores, obtaining the third maximum value of 76.0 % (6.1 % above the baseline value) shown in Table 5.8. The new feature $\boldsymbol{cbF}_{FZ}$ obtained the third maximum value of the counts in the positive bins for accuracy and specificity given in Figure 5.3, obtaining the fourth maximum value of specificity, 75.6 % (5.1 % above the baseline value), given in Table 5.8.

An important result was that the classification scores computed using the new features ***cbA*** and $\boldsymbol{cbA}_{FZ}$ obtained with the ComBat harmonization models, on which the variability introduced by the age variable was conserved, obtained the maximum values of the counts in the negative bins given in Figure 5.3 among all the new features obtained with the ComBat models. This result was also confirmed by the P-values given in Table 5.7 for these features.

Experimental results: Normalization methods

Figure 5.8 shows the classification scores computed with ASD-DiagNet, using the new features obtained from the normalization methods described in Section 5.2.7, on which they are compared to the baseline classification scores given in Table 5.5.

The maximum value of specificity among all the features (see Table 5.8), of 77.0 % (7.5 % above the baseline value), was obtained with the new feature, $\Delta \boldsymbol{avgSite}$, for the sub-sample with 10 sites, which also obtained the maximum counts in the positive bins for specificity (see Figure 5.3). The specificity scores computed with this feature were also greater than ten of the baseline specificity scores given in Figure 5.8. This feature also obtained an accuracy score of 73.8 % (5.1 % above the baseline value), which was among the first five maximum accuracy values given in Table 5.8, and obtained the second maximum counts in the positive bins for accuracy given in Figure 5.3. The experimental results also showed that the feature

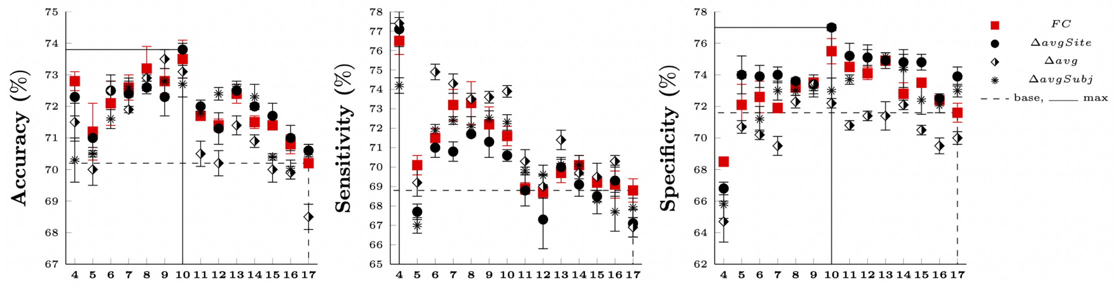


Figure 5.8: Classification scores computed with ASD-DiagNet, using the new features obtained from the normalization methods described in Section 5.2.7, compared with the baseline classification scores (FC) given in Table 5.5. The baseline values for the whole 17 sites are indicated by the dashed line, while the maximum values are indicated by the continuous line.

$\Delta \boldsymbol{avg}$ was among the first five features with the maximum counts in the positive bins for sensitivity given in Figure 5.3, with sensitivity scores greater than eight of the baseline sensitivity scores, obtaining the fifth maximum value of 77.4 % (12.5 % above the baseline value) among the sensitivity values shown in Table 5.8. This feature also obtained the maximum counts in the negative bins for the specificity scores given in Figure 5.3, this result was confirmed by the p-values given in Table 5.7 for this feature.

The results for specificity scores also showed that the feature $\Delta\boldsymbol{avgSubj}$, was among the first six features with the maximum counts in the positive bins for the specificity scores, and among the first five features with the maximum counts in the positive bins for accuracy given in Figure 5.3, respectively.

5.4 Summary

Resting-state functional magnetic resonance imaging (rs-fMRI) is a non-invasive imaging technique widely used in neuroscience to understand the functional connectivity of the human brain. While rs-fMRI multi-site data can help to understand the inner workings of the brain, the data acquisition and processing of this data has many challenges. One of the challenges is the variability of the data associated with different acquisitions sites, and different MRI machines vendors. Other factors such as population heterogeneity among different sites, with variables such as age and gender of the subjects, must also be considered. Given that most of the machine-learning models are developed using these rs-fMRI multi-site data sets, the intrinsic confounding effects can adversely affect the generalizability and reliability of these computational methods, as well as the imposition of upper limits on the classification scores.

The goals of the study presented in this chapter were twofold i) the identification of the phenotypic and imaging variables producing the confounding effects, and ii) to control these confounding effects to maximize the classification scores obtained from the machine learning analysis the rs-fMRI ABIDE multi-site data. To achieve these goals, we proposed a comprehensive approach for controlling the confounding effects on the machine learning analysis of rs-fMRI multi-site data. Our approach consisted of a novel combination of stratification techniques to produce a suitable set of homogeneous sub-samples, as well as the generation of new features for the machine learning analysis through multiple linear regression models, ComBat harmonization models, and normalization methods. The new features obtained with the multiple linear regression models were designed to identify and quantify the effects of phenotypic and imaging variables on the confounding effects. Furthermore, the new features obtained with the ComBat models and the normalization methods were implemented to maximize the classification scores computed with the machine learning analysis performed with our existing state of the art machine-learning models ASD-DiagNet and ASD-SAENet.

The main results obtained with our proposed models and methods were an accuracy of 76.4 %, sensitivity of 82.9 %, and specificity of 77.0 %, which are 8.8 %, 20.5 %, and 7.5 % above the baseline classification scores obtained from the machine learning analysis of the static functional connectivity computed from the ABIDE rs-fMRI multi-site data. These experimental results demonstrated the effectiveness of our proposed approach to quantify the confounding effects of the phenotypic and imaging variables, as well as to maximize the classification scores that were obtained with the proposed statistical models and methods. The codes to implement the algorithms proposed in this chapter are available at https://github.com/pcdslab/ASD-DiagNet-Confounds.

The main conclusion obtained from the comprehensive approach and results presented in this chapter is that the control of the confounding effects, intrinsic to rs-fMRI multisite data, over the machine learning analysis of this type of data, is an essential step towards discovering the functions and structure of the human brain, detecting brain disorders, and defining useful and effective biomarkers for the diagnosis of these disorders. We hope that our approach will be used and improved by the neuroscience research community to maximize the classification scores of the machine learning analysis of rs-fMRI multi-site data.

Our future work will include the use of different sets of rs-fMRI multi-site data, different preprocessing pipelines, as well as, the implementation of data-driven brain parcellations derived from the fMRI data [AKM+18, Mes20] to define the nodes of the functional networks [FEJ+20, FBS22]. We also propose to apply alternative methods to the Pearson correlation to compute the functional connectivity matrices such as mutual information and coherence methods [SMD04, WBQ+14, BS16]. Furthermore, we will explore the application of novel methods for the determination of optimal sub-samples to reduce the confounding effects by using, for example, between-group effect size methods.

CHAPTER 6

ASSESSMENT OF TIME-VARYING FUNCTIONAL CONNECTIVITY IN A MULTI-SITE RS-FMRI DATA FRAMEWORK

In this chapter, we present a preliminary assessment of time-varying functional connectivity in a multi-site rs-fMRI data framework.

6.1 Introduction

At the macroscopic scale, functional connectivity (FC) can be measured as the statistical correlations between rs-fMRI time series recorded at different brain regions (see Section 5.2.1). Static functional connectivity (sFC) is computed by assuming that functional connectivity is constant in time, i.e., computed within the entire rs-fMRI scanning session, capturing instantaneous statistical relationships between brain areas within a scanning session.

The assessment of sFC on multiple studies spanning the last two decades, has increased our knowledge of the functional organization of the human brain, including improvements in the classification of control subjects and subjects with brain disorders [SCKH04, STK05, SR07, vdHSBP08, BB11, Spo12, EMF^{+}19, AS21a].

Considering that the human brain is a complex non-linear dynamic system [BT02], the assumption of static functional connectivity is an important limitation to advancing our knowledge of the dynamic functional brain. Considering this limitation, recent research has evolved to use the concept of time-varying functional connectivity (tvFC), i.e., functional connectivity, measured as the statistical correlations between the rs-fMRI time series recorded at different brain regions, that vary as a function of time. tvFC can be computed within given segments of the rs-fMRI scanning session, shorter than the entire scanning session, capturing instantaneous statistical relationships between brain areas within each given segment of the scanning session.

There is no consensus if the true non-linear dynamics of the functional brain can be obtained from the tvFC computed from rs-fMRI time series [ZFC$^+$14, ADP$^+$14, ADM$^+$17, NVD$^+$17, KEFK$^+$17] , or if the variations in tvFC are due to confound factors like head motion [LSM$^+$17]. Despite this controversy, there are a growing number of studies using tvFC to detect non-linear dynamics of the functional brain [CG10, TVWM$^+$12, HWG$^+$13, ADP$^+$14, HB18, MMO19], and also for improved classification of control and diseased subjects with schizophrenia [SPK$^+$10, RAD$^+$16, FIT$^+$21, GCK$^+$20], autism [FTD$^+$19, ZXS$^+$22, GSC$^+$22], mild cognitive impairment [ZZC$^+$17], bipolarity [RAD$^+$16, DZL$^+$21], stroke [IAV$^+$23], and Alzheimer's disease [JVM$^+$12]. More details about the results and methods related to tvFC studies are described in references [HWA$^+$13, CMPA14, PBVDV17, LKB$^+$20].

In this study, we used functional networks as defined in Section 5.2.2, with the nodes representing brain regions, defined by the cc200 brain atlas. The rs-fMRI time series were obtained from the sites of the ABIDE multi-site data shown in Table 6.1. The rs-fMRI time series of each brain region were segmented using a sliding time window technique [BK86, SPK$^+$10, ADP$^+$14, LVDV15], and then the time-varying functional connectivity (tvFC) was obtained as a time-sequence of static functional connectivity (sFC) values computed for each segment. The computed tvFC represented the time-varying weights of the edges of temporal functional networks. We performed statistical tests to each tvFC representing each ABIDE subject, to determine if the time-variability of each tvFC represents the non-linear dynamics of the functional brain. The test statistics for the null hypothesis used for the statistical tests were obtained from surrogate data [TEL$^+$92] generated from the rs-fMRI time series of each node.

An important factor to consider in tvFC studies is that the sliding time window represents a low-pass filter in the frequency domain, with a cutoff frequency inversely proportional to the window width in seconds, hence the size of window width has an important impact in these studies which should be carefully evaluated [LVDV15, SLK16, ZB15].

Global signal regression is a procedure on which the global average time series of all the voxels in the brain is removed through linear regression from the time series of each individual voxel [MF17], and bandpass filtering, 0.01-0.1 Hz for the ABIDE rs-fMRI data, to remove noise and scanner drift from the rs-fMRI signals usually with frequencies below 0.01 Hz [BSB17, SMM$^+$16], are two standard stages in the preprocessing of ABIDE rs-fMRI data. Some studies have shown that the global signal regression and the bandpass filtering of rs-fMRI data have an impact on the non-linear dynamics of the functional brain network contained in the tvFC values [MF17, LNF17, XSQ$^+$18]. Small head movements during the scanning process of the brain produce noise that can also impact the results of studies related to time-varying functional connectivity [PSP15, CWP$^+$17, PBVDV17].

Considering all the factors described above, we define as the goal of this preliminary study, the assessment of the effects of the width in seconds of the time windows, the bandpass filtering and global signal regression, and the head motion over the non-linear dynamics of the functional brain network contained in the tvFC values obtained from the rs-fMRI data for each subject of the ABIDE sites given in Table 6.1.

6.2 Methods and materials

Table 6.1: International Imaging Sites from preprocessed ABIDE resting state fMRI (http://preprocessed-connectomes-project.org/abide/) used in this chapter [CBC+13].

Sites: California Institute of Technology (Caltech), Carnegie Mellon University (CMU), Kennedy Krieger Institute (KKI), University of Leuven (Leuven), Ludwig Maximilian University (MaxMun), Oregon Health and Science University (OHSU), Institute of Living at Hartford Hospital (Olin), University of Pittsburgh School of Medicine (Pitt), Social Brain Lab (SBL), San Diego State University (SDSU), Stanford University (Stanford), Trinity Center for Health Sciences (Trinity), University California Los Angeles (UCLA),University of Michigan (UM), University of Utah School of Medicine (USM), and Child Study Center, Yale University (Yale).

MRI vendors: General Electric (GE), Phillips(P), Siemens(S)

Site	C	ASD	Subjects	Avg age	Avg FIQ	M/F	MRI
Caltech	18	19	37	27.7 ± 10.4	111.5 ± 11.5	29/8	S
KKI	27	12	39	9.9 ± 1.2	106.8 ± 14.9	29/10	P
Leuven	34	27	61	18.1 ± 5.0	112.2 ± 13.0	54/7	P
MaxMun	24	18	42	27.9 ± 11.2	112.5 ± 11.7	38/4	S
NYU	98	73	171	15.4 ± 6.6	110.3 ± 14.8	136/35	S
OHSU	11	12	23	10.9 ± 1.8	109.8 ± 17.8	23/0	S
Olin	11	14	25	17.1 ± 3.5	114.7 ± 16.7	20/5	S
Pitt	23	22	45	19.2± 6.9	111.4 ± 11.1	38/7	S
SBL	12	14	26	34.9 ± 8.7	109.2 ± 13.6	26/0	P
SDSU	21	12	33	14.6 ± 1.8	110.8 ± 13.3	27/6	GE
Stanford	19	17	36	10.0 ± 1.6	112.3 ± 15.8	28/8	GE
Trinity	23	21	44	17.3 ± 3.4	109.6 ± 13.9	44/0	P
UCLA	39	36	75	13.3 ± 2.2	104.5 ± 11.7	67/8	S
UM	65	48	113	14.5 ± 3.2	108.4 ± 13.4	88/25	GE
USM	23	38	61	23.7 ± 8.3	105.7 ± 18.3	61/0	S
Yale	26	22	48	12.9 ± 2.9	99.9 ± 21.2	34/14	S
TOTAL	474	405	879			742/137	

6.2.1 ABIDE resting fMRI multi-site data

The preprocessed rs-fMRI multi-site data used in this study was obtained from the 16 international imaging sites listed in Table 6.1, publicly available in the ABIDE database, with a total of 474 control and 405 autism subjects [CBC+13, DMYL+14, DMOC+17]. The preprocessing pipeline chosen for this data was the Configurable Pipeline for the Analysis of Connectomes (CPAC).

To assess the effects of the global signal regression (global) and bandpass filtering (filt) stages of the preprocessing ABIDE pipelines over the non-linear dynamics of the functional brain network represented by the tvFC values, we selected four sets of data with the preprocessing strategies: filt-global, nofilt-global, filt-noglobal, nofilt-noglobal. The preprocessing pipeline and the strategies are described in detail in the ABIDE Preprocessed website (http://preprocessed-connectomes-project.org/abide).

6.2.2 Time-varying functional connectivity

To access the effect of the window width (seconds) and the corresponding cutoff frequency of the low-pass filter over the non-linear dynamics of the functional brain network contained in the tvFC values, we selected two groups of time windows. For the first group we selected a constant number of time windows, and for the second we selected constant time widow widths in seconds, both selections applicable to all the ABIDE sites.

Considering that the number of brain volumes of the rs-fMRI data for each ABIDE site is different, we selected for the first group, a constant number of 30- and 60-time windows (nw) applicable to all the sites, with a step (s) of 1 volume between consecutive windows. The widow width in volumes,$w(v)$, was given by

$$w(v) = v - (nw - 1) * s \tag{6.1}$$

where v is the number of brain volumes of the rs-fMRI data. Table 6.2 shows the widths of the time windows in volumes and in seconds, as well as the corresponding cutoff frequencies, $f = 1/w(s)$, which accounts for the low-pass filtering introduced by the time windows and defined in reference [LVDV15, MAS+19]. In this Table, TR is the scan repeat time, i.e., the amount of time needed to record a single brain volume [BSB17, JMS+01], w(vol) is the width of the time window in volumes, and w(s) = w(vol)*TR, the width of the time window in seconds.

Table 6.2: Widths of the time windows and cutoff frequency($f = 1/w(s)$) for the segmentation of the rs-fMRI time-series of the ABIDE sites given in Table 6.1, using 30 and 60 time windows. TR is the scan repeat time, w_{30}(v), w_{60}(v) are the width of the time windows in volumes, and w(s) = w(v)*TR, the width of the time windows in seconds.

Site	v	$w_{30}(v)$	$w_{60}(v)$	$TR(s)$	$w_{30}(s)$	$w_{60}(s)$	$f_{30}(Hz)$	$f_{60}(Hz)$
Caltech	146	117	87	2.0	234	174	0.0043	0.0057
KKI	152	123	93	2.5	308	233	0.0032	0.0043
Leuven	246	217	187	2.0	434	374	0.0023	0.0027
MaxMun	116	87	57	3.0	261	171	0.0038	0.0058
NYU	176	147	117	2.0	294	234	0.0034	0.0043
OHSU	78	49	19	2.5	123	48	0.0081	0.0208
Olin	206	177	147	1.5	266	221	0.0038	0.0045
Pitt	196	167	137	1.5	251	206	0.004	0.0049
SBL	196	167	137	2.2	367	301	0.0027	0.0033
SDSU	176	147	117	2.0	294	234	0.0034	0.0043
Stanford	176	147	117	2.0	294	234	0.0034	0.0043
Trinity	146	117	87	2.0	234	174	0.0043	0.0057
UCLA	116	87	57	3.0	261	171	0.0038	0.0058
UM	296	267	237	2.0	534	474	0.0019	0.0021
USM	236	207	177	2.0	414	354	0.0024	0.0028
Yale	196	167	137	2.0	334	274	0.003	0.0036

Table 6.3: Number of time-windows (nw_{50}, nw_{100}), and window width in volumes ($w_{50}(v)$,$w_{100}(v)$), for the segmentation of the rs-fMRI time-series of the ABIDE sites given in Table 6.1, using window widths of 50 and 100 seconds, respectively.

Site	v	nw_{50}	nw_{100}	$w_{50}(v)$	$w_{100}(v)$
Caltech	146	122	97	25	50
KKI	152	133	113	20	40
Leuven	246	222	197	25	50
MaxMun	116	100	84	17	33
NYU	176	152	127	25	50
OHSU	78	59	39	20	40
Olin	206	174	140	33	67
Pitt	196	164	130	33	67
SBL	196	174	152	23	45
SDSU	176	152	127	25	50
Stanford	176	152	127	25	50
Trinity	146	122	97	25	50
UCLA	116	100	84	17	33
UM	296	272	247	25	50
USM	236	212	183	25	50
Yale	196	172	147	25	50

For the second group of time windows, we selected window widths of 50 and 100 seconds, widths that have been applied in several studies [CG10, SPK+10, HRGCB12, HWG+13, ADP+14, HAM+16, MAS+19]. Table 6.3 shows the number of time-windows (nw_{50}, nw_{100}), and window width in volumes ($w_{50}(v)$,$w_{100}(v)$), for the segmentation of the rs-fMRI time-series of the ABIDE sites given in Table 6.1, using window widths of 50 and 100 seconds, respectively.

Figure 6.1 compares the cutoff frequencies for the low-pass filtering corresponding to the constant number of time windows selected in the first group, and constant width of the time windows in seconds selected in the second group of time windows.

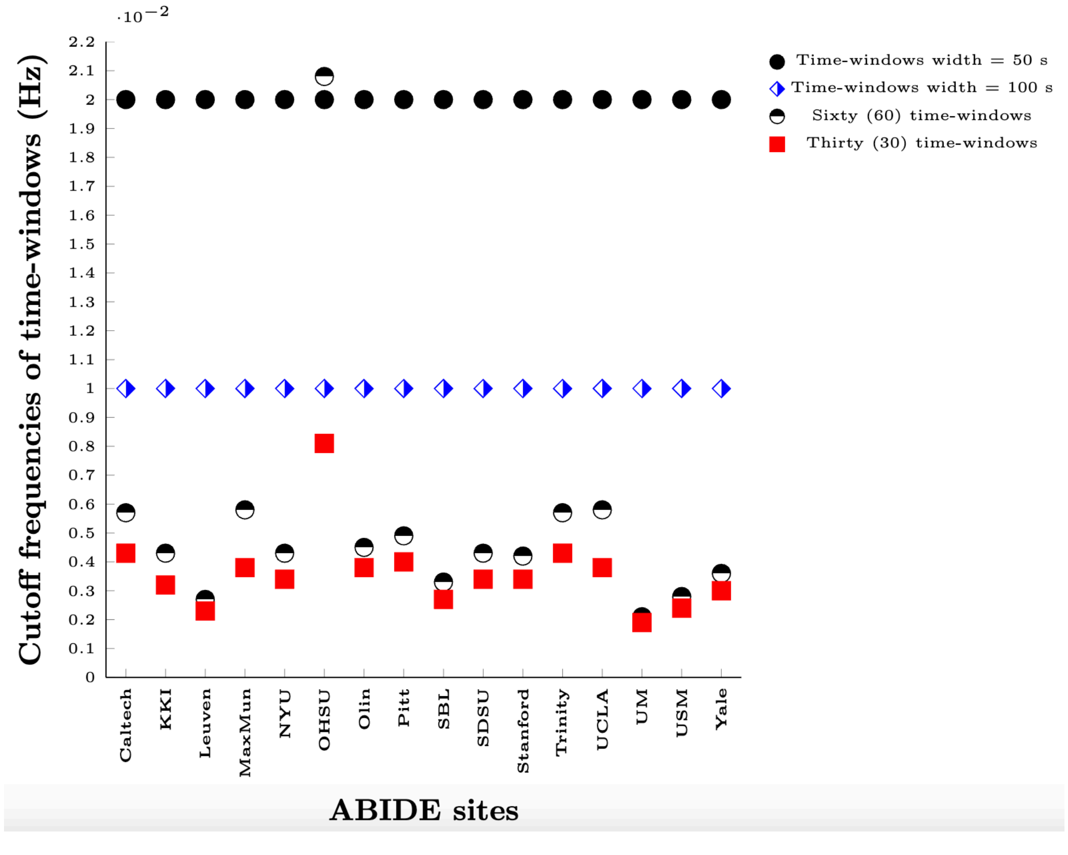


Figure 6.1: Comparison of cutoff frequencies, $f = 1/w(s)$, for the low-pass filtering corresponding to the constant number of time windows, and constant width of the time windows in seconds.

After segmenting the rs-fMRI time series of each node (brain region) of the cc200 brain atlas, we computed the static functional connectivity (sFC) for each segment, as the linear correlation between the time series for all pairs of brain regions, using the Pearson correlation function described in Section 5.2.2. The time-varying functional connectivity (tvFC) was obtained as the time-sequence of the sFC values computed for each segment. The tvFC represented the time-varying weights of the edges of temporal functional networks.

6.2.3 Statistical analysis of the time-varying functional connectivity

In this study, the time-varying functional connectivity computed from rs-fMRI data was considered a sample of tvFC of an unknown population. The claim to be proved is that the variability of the sample tvFC represents non-linear dynamics of the functional connectivity (DFC) of the brain of this population. The classical hypothesis testing formulation of this problem is

$$\begin{aligned} H_0 &: \text{No DFC} \\ H_1 &: \text{DFC} \end{aligned} \tag{6.2}$$

where H_0 is the null hypothesis assumed to be true, and H_1 the alternative hypothesis, representing the claim to be proved [TD00].

Since the null hypothesis H_0 was the claim that the variability of tvFC is due to measurement noise, we needed to obtain the test statistics of the null distribution to verify whether the test statistics of the tvFC, obtained from the rs-fMRI time series, are inside or outside the test statistics of the null distribution. Since the amount of the ABIDE rs-fMRI data is very limited to compute the test statistics of the null distribution, a solution is to generate this distribution from surrogate data [TEL+92]. The surrogate data approach computes the test statistics of the null distribution from a set of constructed (surrogates) data that match the linear properties of the original rs-fMRI time series, but do not possess the nonlinear dynamics which is being tested. In this study, each surrogate was a random copy of the original rs-fMRI time series, on which the static functional connectivity and the frequency spectrum of the rs-fMRI time series were conserved, but any non-linear properties were not conserved. A technique based on the Fourier transform (FT) was applied to generate the required surrogate data [TEL+92].

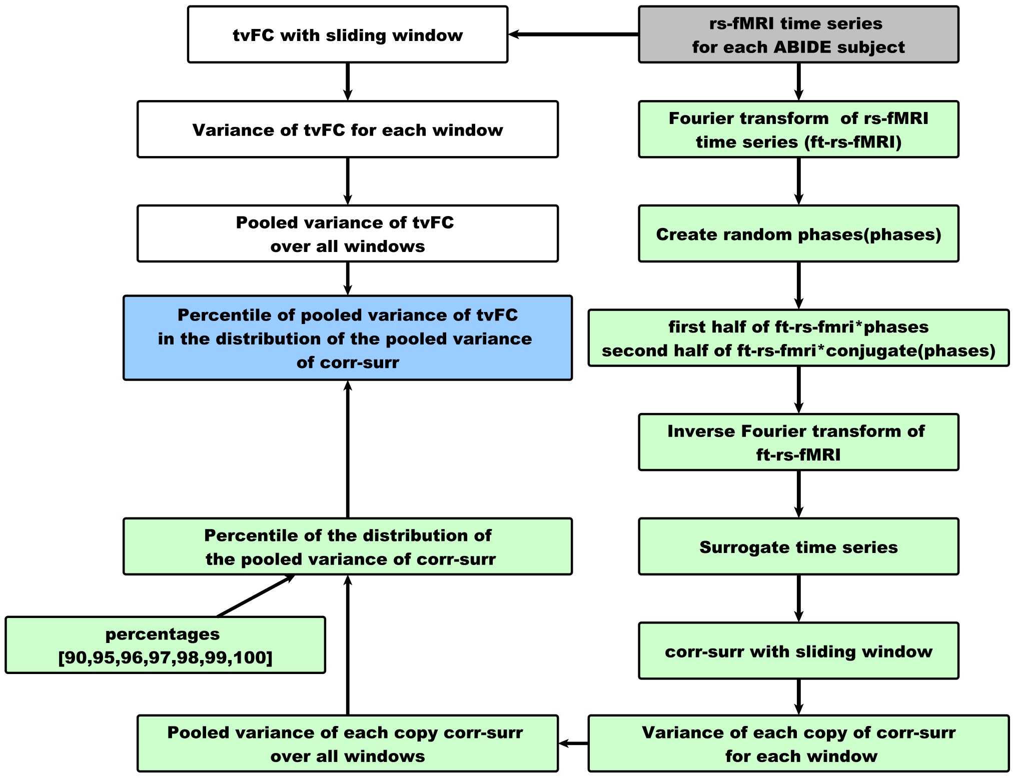

tvFC with sliding window
rs-fMRI time series for each ABIDE subject
Variance of tvFC for each window
Fourier transform of rs-fMRI time series (ft-rs-fMRI)
Pooled variance of tvFC over all windows
Create random phases(phases)
Percentile of pooled variance of tvFC in the distribution of the pooled variance of corr-surr
first half of ft-rs-fmri*phases second half of ft-rs-fmri*conjugate(phases)
Inverse Fourier transform of ft-rs-fMRI
Percentile of the distribution of the pooled variance of corr-surr
Surrogate time series
percentages [90,95,96,97,98,99,100]
corr-surr with sliding window
Pooled variance of each copy corr-surr over all windows
Variance of each copy of corr-surr for each window

Figure 6.2: Workflow to compute the percentile of the pooled variance of tvFC (pvar$_{tvFC}$) in the distribution of the pooled variance (pvar$_{corr-surr}$).

In this study, we used the pooled variance (pvar) over all windows as test statistics. The pooled variance is computed as the average of all the variances of the tvFC components over all windows, namely

$$\text{pvar} = \sum_{i=1}^{i=nw} \frac{\text{var}_i}{nw} \tag{6.3}$$

where nw is the number of time-windows.

Considering the test statistics given by Equation 6.3, we reformulate the hypotheses of Equation 6.2 for each subject as the following one-sided test

$$\begin{aligned} H_0 &: \text{pvar}_{tvFC} = \text{pvar}_{corr-surr} \\ H_1 &: \text{pvar}_{tvFC} > \text{pvar}_{corr-surr} \end{aligned} \tag{6.4}$$

where pvar_{tvFC} is the pooled variance of tvFC, and $\text{pvar}_{corr-surr}$ is the pooled variance of the linear correlation between brain areas computed from the surrogate data.

To test the null hypothesis H_0 in Equation 6.4, we computed the percentile of the pooled variance pvar_{tvFC} in the distribution of the pooled variance of the surrogate data, $\text{pvar}_{corr-surr}$. If the percentile of pvar_{tvFC} is, for example, 95 %, then H_0 is rejected with an observed level of significance or P-value of 0.05.

The workflow given in Figure 6.2 shows the steps needed to compute the percentile of the pooled variance of tvFC (pvar_{tvFC}) in the distribution of the pooled variance ($\text{pvar}_{corr-surr}$), for each ABIDE subject of the sites given in Table 6.1.

The first five steps of the workflow, show the computation of the copies of surrogate time series which were random, due to the multiplication of the Fourier transform of the rs-fMRI time series by random phases, but conserving the linear correlation between brain areas, and the frequency spectrum of the rs-fMRI time series. In the sixth step, we segmented the surrogate data of each node of the cc200 brain atlas, with the time windows given in Table 6.2, and computed the linear correlation between the time series for all pairs of brain regions, for each segment, using the Pearson correlation function described in Section 5.2.2. In the next steps, we computed the pooled variance over all windows (Equation 6.3), the correlation value of each copy of the surrogate data (corr-surr), and the percentiles of this pooled variance using the percentages ([90,95,96,97,98,99,100]), which correspond to P-values of [0.1,0.05,0.04,0.03,0.02,0.01,0.0], respectively.

From the rs-fMRI time series for each ABIDE subject, shown in Figure 6.2, we also computed the pooled variance over all windows (Equation 6.3) of the tvFC obtained from the rs-fMRI time series. Finally, we computed the percentile of the pooled variance of tvFC (pvar$_{tvFC}$) in the distribution of the pooled variance (pvar$_{corr-surr}$), by comparing the pvar$_{tvFC}$ values with the percentile of pvar$_{corr-surr}$, if a pvar$_{tvFC}$ value was, for example, equal or greater than the percentile 95 and less than the percentile 94 of pvar$_{corr-surr}$, then the percentile of the pvar$_{tvFC}$ value was equal to 95, and therefore, H_0 was rejected with a P-value of 0.05. As experimental result, we selected the minimum P-value for the percentile of each pvar$_{tvFC}$ value of each subject.

6.2.4 Assessment of the effect of head motion over the variability of tvFC

One of the measures of subject head motion used in the quality assessment (QA) of the preprocessed ABIDE rs-fMRI data is the mean framewise displacement (FD) which compares the motion between the current and previous volumes of the scanning session [PSP15]. There are two other measures derived from the FD measure: The number of FD greater than 0.2 mm, i.e., the number of volumes with displacement greater than 2 mm, and the percentage of FD greater than 0.2 mm, i.e., the percentage of volumes with displacement greater than 2 mm. A detailed description of these measures is given on the ABIDE website: http://preprocessed-connectomes-project.org/abide/quality-assessment.html.

In this study, we used the percentage of FD greater than 0.2 mm, as a metric to assess the effect of head motion over the non-linear dynamics of the functional brain network contained in the tvFC values. We created a histogram by counting the number of subjects for which the null hypothesis was rejected, and which have a head motion less than a given range of values of percentages of FD.

6.3 Results

In this study, we performed experiments to assess the impact of the time-window width, the filtering and global signal regression, and the head motion over the non-linear dynamics of the functional brain network contained in the tvFC values obtained from the rs-fMRI data for each subject of the ABIDE sites given in Table 6.1. For these experiments, we generated 250 copies of the surrogate data and computed the percentile of the pooled variance of tvFC (pvar$_{tvFC}$) in the distribution of the pooled variances (pvar$_{corr-surr}$) of all the surrogate copies, using the steps shown in Figure 6.2. We computed the pooled variance over all time windows (Equation 6.3), the correlation value of each copy of the surrogate data (corr-surr), and the percentiles of this pooled variance using the percentages ([90,95,96,97,98,99,100]), which correspond to P-values of [0.1,0.05,0.04,0.03,0.02,0.01,0.0], respectively.

These experimental results were obtained for all the 879 ABIDE subjects summarized in Table 6.1, using the strategies for the ABIDE pipelines described in Section 6.2.1.

6.3.1 Experimental results

This section presents the experimental results obtained using: 1) a constant number of time windows, 2) time windows with constant width, and 3) the percentage of FD greater than 0.2 mm to assess the effect of head motion.

Constant number of time windows: Experimental results

This section presents the experimental results obtained using 30 and 60-time windows for all ABIDE subjects, as described in Section 6.2.2.

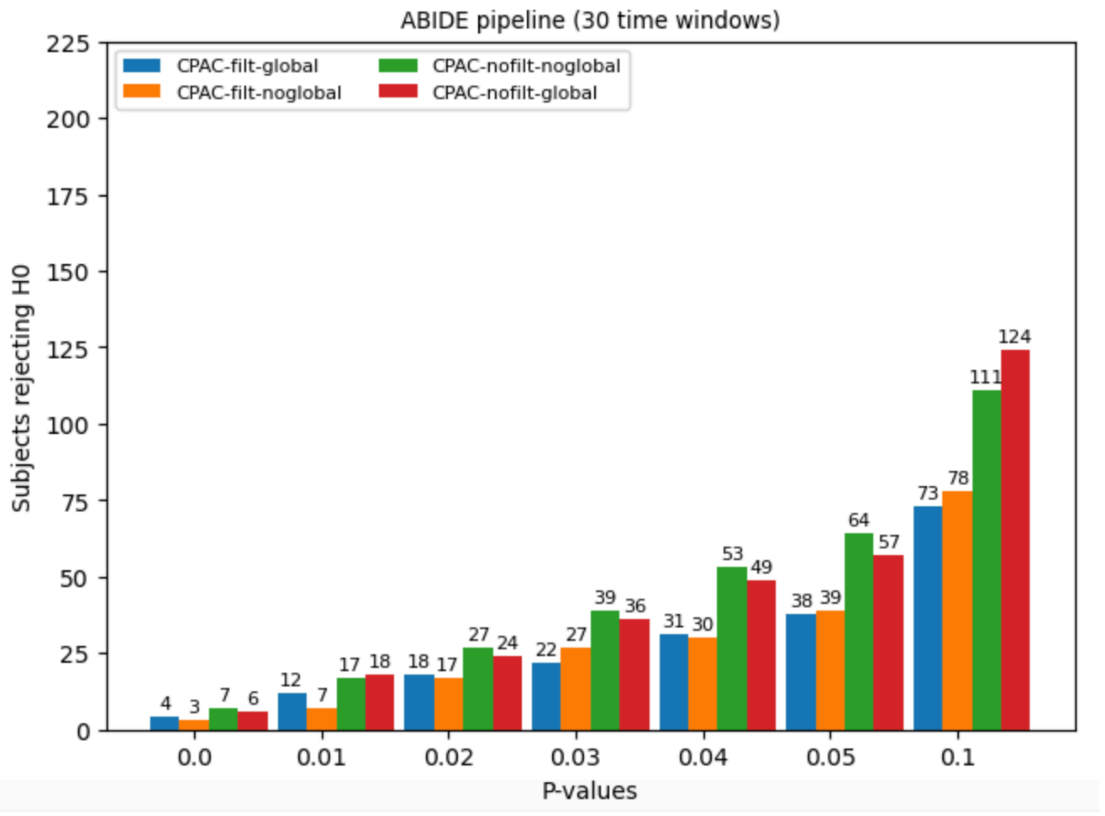


Figure 6.3: Comparison between the number of subjects for which H_0 was rejected with P-values equal to or less than 0.1, obtained for 30 time windows, and for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1.

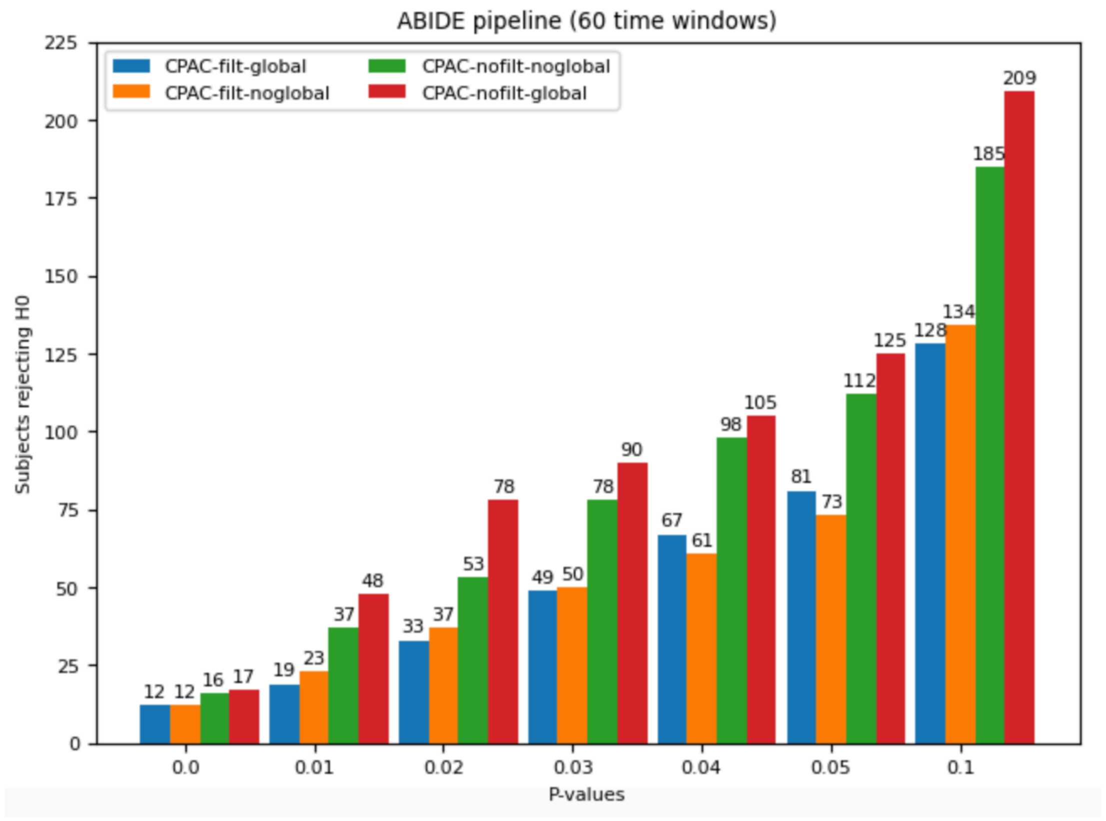


Figure 6.4: Comparison between the number of subjects for which H_0 was rejected with P-values equal to or less than 0.1, obtained for 60-time windows, and for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1.

Figures 6.3 and 6.4 present a comparison between the number of subjects for which H_0 was rejected with P-values equal or less than 0.1, obtained for 30 and 60-time windows respectively, and for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1. These results showed that the two greatest number of subjects for which the null hypothesis H_0 was rejected, were obtained for the no-filtering strategies (nofilt-global and nofilt-noglobal) of the CPAC pipeline, respectively. We also showed that due to the greater cutoff frequencies corresponding to 60-time windows compared to the frequencies obtained for 30-time windows (see Figure 6.1), the maximum number of subjects rejecting H_0 was 209 subjects for 60-time windows (24 % of all ABIDE subjects in Table 6.1), 69 % greater than the 124 subjects obtained for 30-time windows. These maximum results were obtained for the nofilt-global strategy. These results also allowed us to that for these maximum numbers, the null hypothesis H_0, i.e., the claim that the variability of tvFC is due to measurement noise, was rejected with a P-value of 0.1, and, therefore there is sufficient evidence, at a P-value of 0.1, to support the claim that the variability of tvFC represents non-linear dynamics of the functional connectivity (DFC) of the brain.

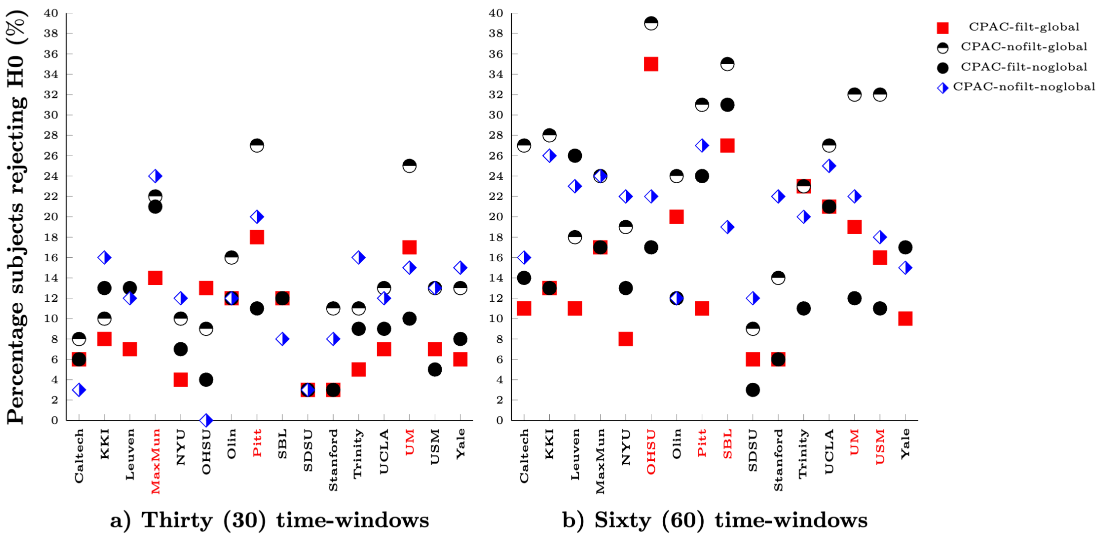


Figure 6.5: Comparison between the percentage of subjects for which H_0 was rejected per ABIDE site, for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1, and for 30 and 60-time windows. The sites in red are those with the maximum number of subjects for which H_0 was rejected with P-values equal to or less than 0.1.

Figure 6.5 presents a comparison between the percentage of subjects for which H_0 was rejected per ABIDE site, for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1, and for 30 and 60-time windows. These results were highly dependent on the site, with the sites in red (OHSU, Pitt, SBL, UM, and USM), obtaining the maximum percentages of subjects for which H_0 was rejected with P-values equal to or less than 0.1. An important result was that the maximum cutoff frequency of 0.0208 Hz (see Table 6.1) corresponded to the OHSU site that also obtained the maximum percentage (39 %) of subjects for which $H0$ was rejected. These results also showed that the maximum percentage of subjects rejecting H_0 corresponded to the 60-time windows, due to the greater cutoff frequencies compared to the corresponding frequencies of the 30-time windows case. We also confirmed that the maximum results were obtained for the no-filtering strategies (nofilt-global and nofilt-noglobal) of the CPAC pipeline and that these results were consistent for all the P-values and all the sites, respectively.

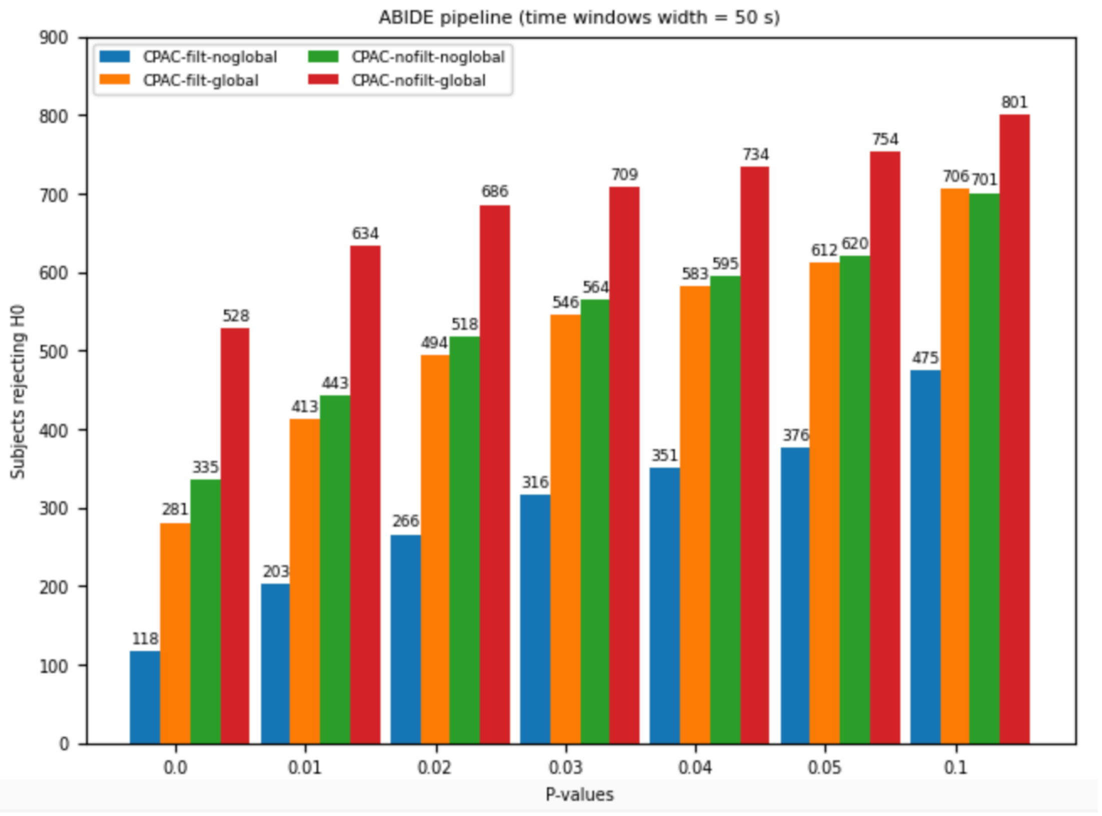


Figure 6.6: Comparison between the number of subjects for which H_0 was rejected with P-values equal to or less than 0.1, obtained for time windows with a width size of 50 seconds, and for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1.

Time windows with constant width sizes: Experimental results

This section presents the experimental results obtained using time windows with width sizes of 50 and 100 seconds for all ABIDE subjects, as described in Section 6.2.2.

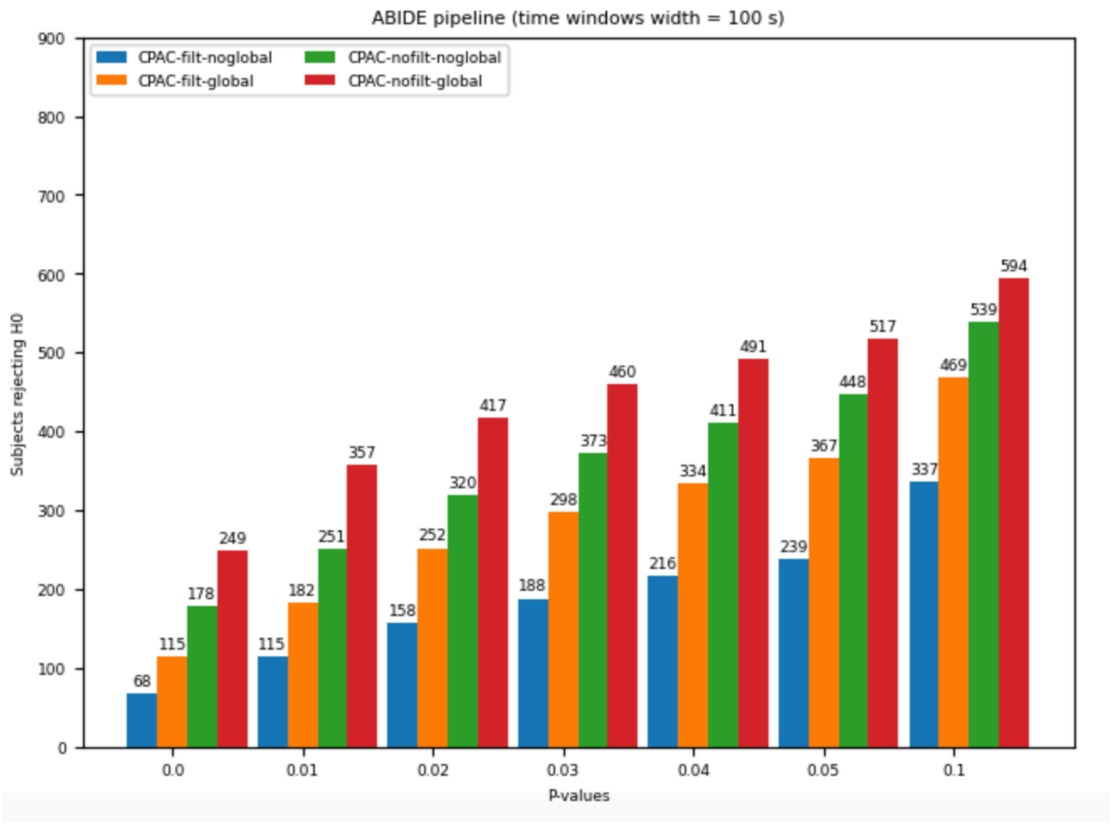


Figure 6.7: Comparison between the number of subjects for which H_0 was rejected with P-values equal to or less than 0.1, obtained for time windows with a width size of 100 seconds, and for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1.

Figures 6.6 and 6.7 present a comparison between the number of subjects for which H_0 was rejected with P-values equal or less than 0.1, obtained for time windows with width sizes of 50 and 100 seconds respectively, and for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1. These results showed that the two greatest number of subjects for which the null hypothesis H_0 was rejected, were obtained for the no-filtering strategies (nofilt-global and nofilt-noglobal) of the CPAC pipeline, respectively. We also showed that due to the greater cutoff frequencies corresponding to time windows with a width size of 50 seconds compared to the frequencies obtained for a width size of 100 seconds (see Figure 6.1), the maximum number of subjects rejecting H_0 was 801 subjects for a width size of 50 seconds (91 % of all ABIDE subjects in Table 6.1), 35 % greater than the 594 subjects obtained for a width size of 100 seconds. These maximum results were obtained for the nofilt-global strategy. These results also allowed us to conclude that for these maximum numbers, the null hypothesis H_0, i.e., the claim that the variability of tvFC is due to measurement noise, was rejected with a P-value of 0.1, and, therefore there is sufficient evidence, at a P-value of 0.1, to support the claim that the variability of tvFC represents non-linear dynamics of the functional connectivity (DFC) of the brain.

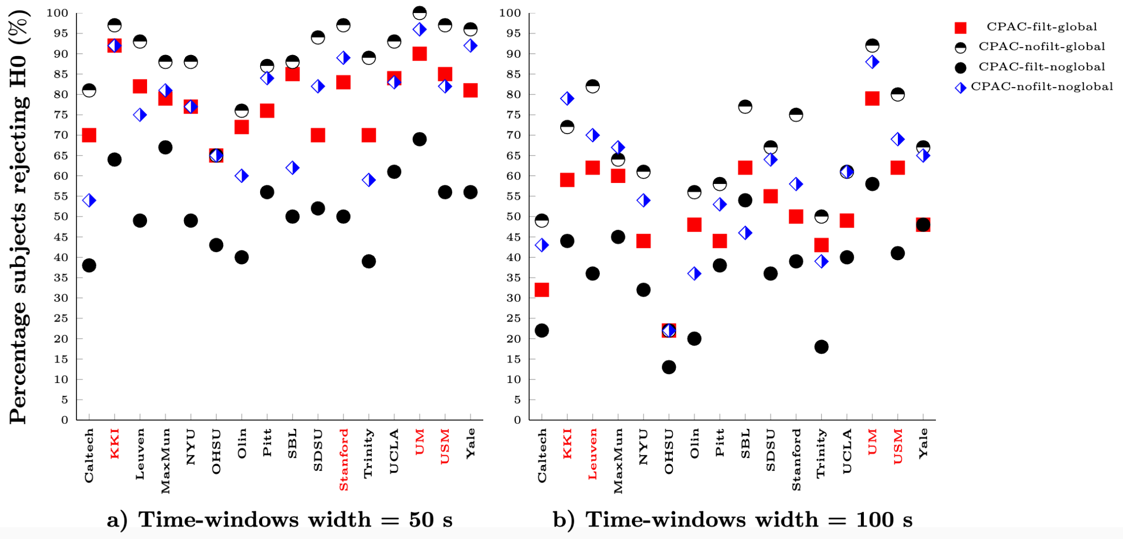


Figure 6.8: Comparison between the percentage of subjects for which H_0 was rejected per ABIDE site, for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1, and for time windows with width sizes of 50 and 100 seconds. The sites in red are those with the maximum number of subjects for which H_0 was rejected with P-values equal to or less than 0.1.

Figure 6.8 presents a comparison between the percentage of subjects for which H_0 was rejected per ABIDE site, for the four strategies of the CPAC ABIDE pipeline given in Section 6.2.1, and for time windows with width sizes of 50 and 100 seconds. These results were highly dependent on the site, with the sites in red (KKI, Stanford, UM, and USM), obtaining the maximum percentages of subjects for which H_0 was rejected with P-values equal to or less than 0.1. These results also showed that due to the greater cutoff frequencies obtained for the width size of 50 seconds compared to the corresponding frequencies of the width size of 100 seconds, the maximum percentages of subjects rejecting H_0 corresponded to the width size of 50 seconds. Furthermore, we demonstrated that the maximum percentages were obtained for the nofilt-global strategy of the CPAC pipeline and that these results were consistent for all the P-values and all the sites, respectively.

Effect of head motion: Experimental results

This section presents the experimental results obtained to assess the effect of head motion over the non-linear dynamics of the functional brain network contained in the tvFC values using the percentages of the mean framewise displacement (FD) greater than 0.2 mm for all the ABIDE subjects. We created a histogram by counting the cumulative percentage of subjects for which the null hypothesis was rejected, and which have a head motion less than a given range of values of percentages of FD. The range of percentages of FD were $[0.0, 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0, 27.0]$, on which, for example, 0.0 corresponds to the cumulative percentage of subjects for which the null hypothesis was rejected with percentages of FD less than 3.0 %, and 3.0 to the cumulative percentage of subjects with percentages of FD less than 6.0%, and so forth.

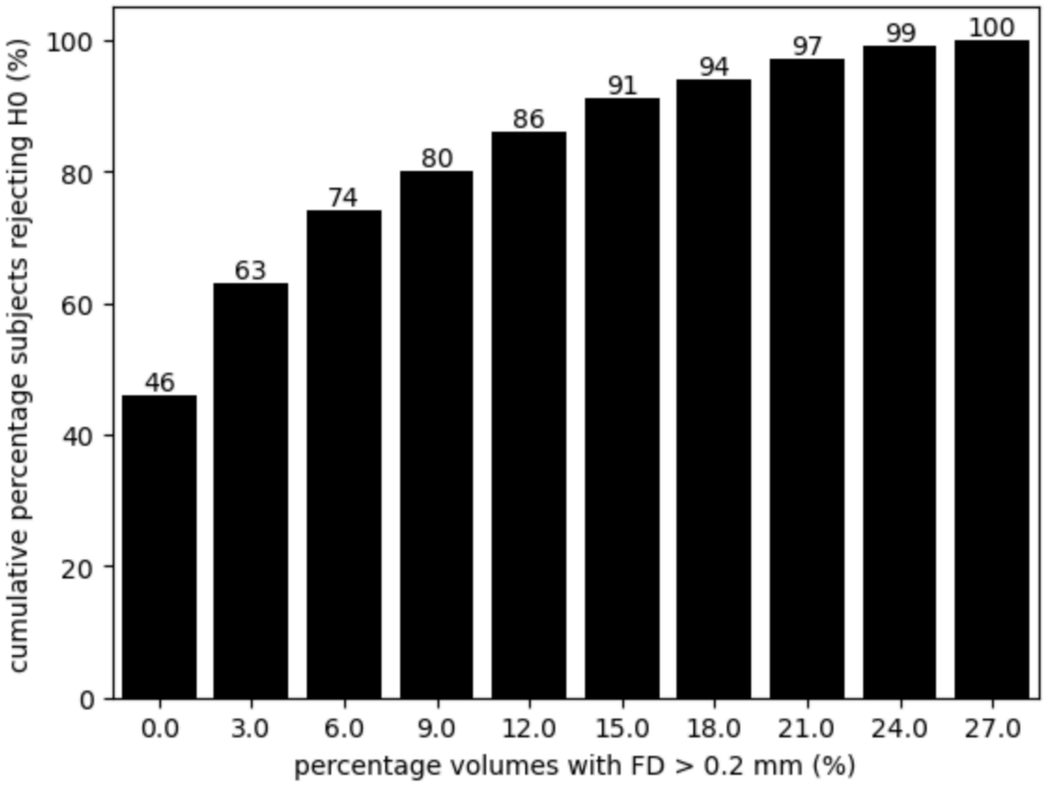


Figure 6.9: Histogram showing the ranges of percentages of FD for the cumulative percentage of subjects rejecting $H0$, for the nofilt-global strategy of the CPAC pipeline, and for time windows with a constant width of 50 seconds.

Figure 6.9 shows the ranges of percentages of FD for the cumulative percentage of subjects rejecting $H0$. The histogram was obtained for the nofilt-global strategy of the CPAC pipeline and for time windows with a constant width of 50 seconds. This histogram showed that for about 45 % of the subjects rejecting $H0$, the percentage of FD greater than 0.2 mm was less than 3 %, and for 80 % of the subjects this percentage was less than 12 %. These results allowed us to conclude that the effect of the head motion over the non-linear dynamics of the functional brain network contained in the tvFC values had an upper bound determined by the percentages of FD greater than 0.2 mm. This upper bound, for example, was less than 12 % for 80 % of the subjects rejecting $H0$, hence, for these subjects, a high percentage of the variability of the tvFC values represented the non-linear dynamics of the functional brain network, with the corresponding levels of significance determined by the P-values given in Section 6.3.1.

6.4 Summary

Considering that the human brain is a complex non-linear dynamic system [BT02], the assumption of static functional connectivity is an important limitation to advancing our knowledge of the dynamic functional brain. Considering this limitation, recent research has evolved to use the concept of time-varying functional connectivity (tvFC), i.e., functional connectivity, measured as the statistical correlations between rs-fMRI time series recorded at different brain regions, that vary as a function of time.

In this study, we used functional networks with the nodes representing brain regions. The rs-fMRI time series of each brain region was segmented using a sliding time-window technique, then the time-varying functional connectivity (tvFC) was obtained as a time-sequence of static functional connectivity (sFC) values computed for each segment. We performed statistical tests on the tvFC for each ABIDE subject, to determine if the time-variability of each tvFC represents non-linear dynamics of the functional brain. The test statistics for the null hypothesis used for the statistical tests were obtained from surrogate data generated from the rs-fMRI time series of each node.

The goal of this preliminary study was the assessment of the effects of the width in seconds of the time windows, the bandpass filtering and global signal regression, and the head motion over the non-linear dynamics of the functional brain network contained in the tvFC values obtained from the ABIDE rs-fMRI data.

Our experimental results showed that the two greatest number of subjects for which the null hypothesis H_0 was rejected, were obtained for the no-filtering strategies (nofilt-global and nofilt-noglobal) of the CPAC pipeline, respectively. We also showed that due to the greater cutoff frequencies corresponding to time windows with a width size of 50 seconds compared to the frequencies obtained for a width size of 100 seconds, the maximum number of subjects rejecting H_0 was 801 subjects for a width size of 50 seconds (91 % of all ABIDE subjects in Table 6.1), 35 % greater than the 594 subjects obtained for a width size of 100 seconds. These maximum results were obtained for the nofilt-global strategy of the CPAC pipeline. Furthermore, these results also allowed us to conclude that for these maximum numbers, the null hypothesis H_0, i.e., the claim that the variability of tvFC is due to measurement noise, was rejected with a P-value of 0.1, and, therefore there is sufficient evidence, at a P-value of 0.1, to support the claim that the variability of tvFC represents non-linear dynamics of the functional connectivity (DFC) of the brain. Finally, we concluded that the effect of the head motion over the non-linear dynamics of the functional brain network contained in the tvFC values had an upper bound determined by the percentages of FD greater than 0.2 mm. This upper bound, for example, was less than 12 % for 80 % of the subjects rejecting $H0$, hence, for these subjects, a high percentage of the variability of the tvFC values represented the non-linear dynamics of the functional brain network, with the corresponding levels of significance determined by the P-values.

Our future work will include the use of different sets of rs-fMRI multi-site data, different preprocessing pipelines, as well as, the implementation of data-driven brain parcellations derived from the fMRI data [AKM$^+$18, Mes20]. We also propose to apply alternative methods to the Pearson correlation to compute the functional connectivity such as mutual information and coherence methods [SMD04, WBQ$^+$14, BS16], as well as additional methods for the definition of the nodes of the functional networks [FEJ$^+$20, FBS22], such as data-driven weighted brain parcellations, based on independent component analysis (ICA), which has been used as an alternative to a priory brain parcellations (such as cc200) in the analysis of functional connectivity [BS04, Shl14, DF13, DFS$^+$20]. Furthermore, we will explore the application of novel methods for the determination of optimal sub-samples to reduce the confounding effects by using, for example, between-group effect size methods.

CHAPTER 7

CURRENT AND FUTURE WORK

Modern neuroscience research is mainly focused on understanding the function and structure of the complex dynamic of the human brain. The continuously increasing availability of human brain imaging data, and the development of advanced computational and mathematical techniques for the analysis of complex networks, have propelled important research efforts to improve the analysis and modeling of this data, to increase our understanding of the dynamics of the human brain [KSBB18]. However, this research approach is mainly descriptive and has limitations in explaining the dynamics of the different spatial scales of the human brain [GB14, HB18]. To overcome this limitation, generative dynamic models of functional connectivity have been proposed and developed by applying the mathematical models and theories provided by modern dynamic system analysis [Bre04, BS05, Bre17, CKD17]. One of the main goals of these research efforts is to develop dynamic models of the neuronal activity of the brain and understand how these models may provide explanations of the non-linear variability of imaging data [HB18]. An ambitious goal of current neuroscience research is to develop a comprehensive non-linear model that will eventually provide the association between the dynamic physical brain and the imaging data [Bre17].

A very active area of neuroscience research is related to brain network communication, strongly linked to the dynamics of the human brain. The main goal is to understand the mechanisms through which the neuronal components at the different spatial scales of the brain, communicate and exchange information to perform the complex functions of the human brain, a comprehensive review of the challenges, methods, and future developments of this area of research are given in reference [SSZ23].

We propose as the future work of this dissertation to develop and implement a comprehensive approach to determine to which extent the time-variability of the time-varying functional connectivity, computed from rs-fMRI multi-site data, represents non-linear dynamics of the functional brain. We also propose to use the time-varying functional connectivity as features of machine learning models for the diagnosis of brain disorders.

Furthermore, we propose the development and implementation of high-performance algorithms to compute the mathematical models needed to understand the dynamic and communication networks of the human brain. This contribution may help to provide the link between the dynamic physical brain and the imaging data, a promising and state-of-the-art area of research.

Another research project which naturally emerges from this dissertation, is the implementation of a comprehensive library of high-performance algorithms for computing graphs metrics applicable to the analysis of human brain networks, using modern technologies like multiple GPUs.

BIBLIOGRAPHY

[AAMS23] Oswaldo Artiles, Zeina Al Masry, and Fahad Saeed. Confounding effects on the performance of machine learning analysis of static functional connectivity computed from rs-fmri multi-site data. *Neuroinformatics*, pages 1–18, 2023.

[ACD$^+$20] Mohsen Aznaveh, Jinhao Chen, Timothy A Davis, Bálint Hegyi, Scott P Kolodziej, Timothy G Mattson, and Gábor Szárnyas. Parallel graphblas with openmp. In *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*, pages 138–148. SIAM, 2020.

[ACG$^+$09] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[ADM$^+$17] Anees Abrol, Eswar Damaraju, Robyn L Miller, Julia M Stephen, Eric D Claus, Andrew R Mayer, and Vince D Calhoun. Replicability of time-varying connectivity patterns in large resting state fmri samples. *Neuroimage*, 163:160–176, 2017.

[ADP$^+$14] Elena A Allen, Eswar Damaraju, Sergey M Plis, Erik B Erhardt, Tom Eichele, and Vince D Calhoun. Tracking whole-brain connectivity dynamics in the resting state. *Cerebral cortex*, 24(3):663–676, 2014.

[AGHP89] AM Aertsen, GL Gerstein, MK Habib, and GwtcoPGK Palm. Dynamics of neuronal firing correlation: modulation of" effective connectivity". *Journal of neurophysiology*, 61(5):900–917, 1989.

[AKM$^+$18] Salim Arslan, Sofia Ira Ktena, Antonios Makropoulos, Emma C. Robinson, Daniel Rueckert, and Sarah Parisot. Human brain mapping: A systematic comparison of parcellation methods for the human cerebral cortex. *NeuroImage*, 170:5–30, 2018. Segmenting the Brain.

[AMDM$^+$17] Alexandre Abraham, Michael P Milham, Adriana Di Martino, R Cameron Craddock, Dimitris Samaras, Bertrand Thirion, and Gael Varoquaux. Deriving reproducible biomarkers from multi-site resting-state data: An autism-based example. *NeuroImage*, 147:736–745, 2017.

[AMR$^+$17] Hyeong Su An, Won-Jin Moon, Jae-Kyun Ryu, Ju Yeon Park, Won Sung Yun, Jin Woo Choi, Geon-Ho Jahng, and Jang-Yeon Park. Inter-vender and test-retest reliabilities of resting-state functional magnetic resonance imaging: Implications for multi-center imaging studies. *Magnetic resonance imaging*, 44:125–130, 2017.

[AS19] Oswaldo Artiles and Fahad Saeed. Gpu-sfft: A gpu based parallel algorithm for computing the sparse fast fourier transform (sfft) of k-sparse signals. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3303–3311. IEEE, 2019.

[AS21a] Fahad Almuqhim and Fahad Saeed. Asd-saenet: A sparse autoencoder, and deep-neural network model for detecting autism spectrum disorder (asd) using fmri data. *Frontiers in Computational Neuroscience*, 15:27, 2021.

[AS21b] Oswaldo Artiles and Fahad Saeed. A multi-factorial assessment of functional human autistic spectrum brain network analysis. In *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 3526–3531. IEEE, 2021.

[AS21c] Oswaldo Artiles and Fahad Saeed. Turbobc: A memory efficient and scalable gpu based betweenness centrality algorithm in the language of linear algebra. In *50th International Conference on Parallel Processing Workshop*, pages 1–10, 2021.

[AS21d] Oswaldo Artiles and Fahad Saeed. Turbobfs: Gpu based breadth-first search (bfs) algorithms in the language of linear algebra. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 520–528. IEEE, 2021.

[BAP12] Scott Beamer, Krste Asanovic, and David Patterson. Direction-optimizing breadth-first search. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10. IEEE, 2012.

[Bav48] Alex Bavelas. A mathematical model for group structures. *Applied Anthropology*, 7(3):16–30, 1948.

[BB11] Edward T. Bullmore and Danielle S. Bassett. Brain graphs: Graphical models of the human brain connectome. *Annual Review of Clinical Psychology*, 7(1):113–140, 2011. PMID: 21128784.

[BBAP13] Scott Beamer, Aydin Buluc, Krste Asanovic, and David Patterson. Distributed memory breadth-first search revisited: Enabling bottom-up search. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 1618–1627. IEEE, 2013.

[BCVO+20] AmanPreet Badhwar, Yannik Collin-Verreault, Pierre Orban, Sebastian Urchs, Isabelle Chouinard, Jacob Vogel, Olivier Potvin, Simon Duchesne, and Pierre Bellec. Multivariate consistency of resting-state fmri connectivity maps acquired on a single individual over 2.5 years, 13 sites and 3 vendors. *NeuroImage*, 205:116210, 2020.

[BFG+06] David A Bader, John Feo, John Gilbert, Jeremy Kepner, David Koester, Eugene Loh, Kamesh Madduri, Bill Mann, and Theresa Meuse. Hpcs scalable synthetic compact applications 2 graph analysis. *SSCA*, 2:v2, 2006.

[BG08] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication in cuda. Technical Report NVR-2008-004, NVIDIA, 2008.

[BH12] Nathan Bell and Jared Hoberock. Thrust: A productivity-oriented library for cuda. In *GPU computing gems Jade edition*, pages 359–371. Elsevier, 2012.

[BK86] David S Broomhead and Gregory P King. Extracting qualitative dynamics from experimental data. *Physica D: Nonlinear Phenomena*, 20(2-3):217–236, 1986.

[BMM+17] Aydin Buluç, Tim Mattson, Scott McMillan, Jose Moreira, and Carl Yang. Design of the graphblas api for c. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 643–652, New Orleans, Lousianna, 2017. IEEE Press.

[BMP+13] Rasmus M Birn, Erin K Molloy, Rémi Patriat, Taurean Parker, Timothy B Meier, Gregory R Kirk, Veena A Nair, M Elizabeth Meyerand, and Vivek Prabhakaran. The effect of scan length on the reliability of resting-state fmri connectivity estimates. *Neuroimage*, 83:550–558, 2013.

[BMS+11] Gregory G Brown, Daniel H Mathalon, Hal Stern, Judith Ford, Bryon Mueller, Douglas N Greve, Gregory McCarthy, James Voyvodic, Gary

Glover, Michele Diaz, et al. Multisite reliability of cognitive bold data. *Neuroimage*, 54(3):2163–2175, 2011.

[BMZ⁺10] Bharat B Biswal, Maarten Mennes, Xi-Nian Zuo, Suril Gohel, Clare Kelly, Steve M Smith, Christian F Beckmann, Jonathan S Adelstein, Randy L Buckner, Stan Colcombe, et al. Toward discovery science of human brain function. *Proceedings of the National Academy of Sciences*, 107(10):4734–4739, 2010.

[BPP⁺22] Oualid Benkarim, Casey Paquola, Bo-yong Park, Valeria Kebets, Seok-Jun Hong, Reinder Vos de Wael, Shaoshi Zhang, BT Thomas Yeo, Michael Eickenberg, Tian Ge, et al. Population heterogeneity in clinical cohorts affects the predictive accuracy of brain imaging. *PLoS biology*, 20(4):e3001627, 2022.

[BPS13] Bonnie Berger, Jian Peng, and Mona Singh. Computational solutions for omics data. *Nature Reviews Genetics*, 14(5):333–346, 2013.

[Bra08] Ulrik Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2):136–145, 2008.

[Bre04] Michael Breakspear. "dynamic" connectivity in neural systems. *Neuroinformatics*, 2(2):205–224, 2004.

[Bre17] Michael Breakspear. Dynamic models of large-scale brain activity. *Nature neuroscience*, 20(3):340–352, 2017.

[BS04] Christian F Beckmann and Stephen M Smith. Probabilistic independent component analysis for functional magnetic resonance imaging. *IEEE transactions on medical imaging*, 23(2):137–152, 2004.

[BS05] Michael Breakspear and Cornelis J Stam. Dynamics of a neural system with a multiscale architecture. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1457):1051–1074, 2005.

[BS09] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186–198, 2009.

[BS16] André M Bastos and Jan-Mathijs Schoffelen. A tutorial review of functional connectivity analysis methods and their interpretational pitfalls. *Frontiers in systems neuroscience*, 9:175, 2016.

[BS17] Danielle S Bassett and Olaf Sporns. Network neuroscience. *Nature Neuroscience*, 20(3):353–364, 2017.

[BSB17] Janine Bijsterbosch, Stephen M Smith, and Christian F Beckmann. *An introduction to resting state fMRI functional connectivity.* Oxford University Press, 2017.

[BSV$^+$07] Katy Börner, Soma Sanyal, Alessandro Vespignani, et al. Network science. *Annu. rev. inf. sci. technol.*, 41(1):537–607, 2007.

[BT02] M Breakspear and JR Terry. Nonlinear interdependence in neural systems: motivation, theory, and relevance. *International Journal of Neuroscience*, 112(10):1263–1284, 2002.

[BWB$^+$09] Jason W Bohland, Caizhi Wu, Helen Barbas, Hemant Bokil, Mihail Bota, Hans C Breiter, Hollis T Cline, John C Doyle, Peter J Freed, Ralph J Greenspan, et al. A proposal for a coordinated effort for the determination of brainwide neuroanatomical connectivity in model organisms at a mesoscopic scale. *PLoS computational biology*, 5(3):e1000334, 2009.

[BZYHH95] Bharat Biswal, F Zerrin Yetkin, Victor M Haughton, and James S Hyde. Functional connectivity in the motor cortex of resting human brain using echo-planar mri. *Magnetic resonance in medicine*, 34(4):537–541, 1995.

[CBC$^+$13] Cameron Craddock, Yassine Benhajali, Carlton Chu, Francois Chouinard, Alan Evans, András Jakab, Budhachandra Singh Khundrakpam, John David Lewis, Qingyang Li, Michael Milham, et al. The neuro bureau preprocessing initiative: open sharing of preprocessed neuroimaging data and derivatives. *Frontiers in Neuroinformatics*, 7:3, 2013.

[CF14] Gregory W Corder and Dale I Foreman. *Nonparametric statistics: A step-by-step approach.* John Wiley and Sons, Hoboken, New Jersey., 2014.

[CG10] Catie Chang and Gary H Glover. Time–frequency dynamics of resting-state brain connectivity measured with fmri. *Neuroimage*, 50(1):81–98, 2010.

[CJ99] Robert W Cox and Andrzej Jesmanowicz. Real-time 3d image registration for functional mri. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 42(6):1014–1018, 1999.

[CJ15] Etienne Combrisson and Karim Jerbi. Exceeding chance level by chance: The caveat of theoretical chance levels in brain signal classification and statistical assessment of decoding accuracy. *Journal of neuroscience methods*, 250:126–136, 2015.

[CJHI$^+$12] R Cameron Craddock, G Andrew James, Paul E Holtzheimer III, Xiaoping P Hu, and Helen S Mayberg. A whole brain fmri atlas generated via spatially constrained spectral clustering. *Human brain mapping*, 33(8):1914–1928, 2012.

[CJY$^+$13] R Cameron Craddock, Saad Jbabdi, Chao-Gan Yan, Joshua T Vogelstein, F Xavier Castellanos, Adriana Di Martino, Clare Kelly, Keith Heberlein, Stan Colcombe, and Michael P Milham. Imaging human connectomes at the macroscale. *Nature methods*, 10(6):524–539, 2013.

[CKD17] Joana Cabral, Morten L. Kringelbach, and Gustavo Deco. Functional connectivity dynamically evolves on multiple time-scales over a static structural connectome: Models and mechanisms. *NeuroImage*, 160:84–96, 2017. Functional Architecture of the Brain.

[CKJ$^+$15] Colleen P Chen, Christopher L Keown, Afrooz Jahedi, Aarti Nair, Mark E Pflieger, Barbara A Bailey, and Ralph-Axel Müller. Diagnostic classification of intrinsic functional connectivity highlights somatosensory, default mode, and visual regions in autism. *NeuroImage: Clinical*, 8:238–245, 2015.

[CKM13] Colleen Pam Chen, Christopher Lee Keown, and Ralph-Axel Müller. Towards understanding autism risk factors: a classification of brain images with support vector machines. *Int. J. Semantic Comput.*, 7(2):205, 2013.

[CLC$^+$14] Jiayu Chen, Jingyu Liu, Vince D Calhoun, Alejandro Arias-Vasquez, Marcel P Zwiers, Cota Navin Gupta, Barbara Franke, and Jessica A Turner. Exploration of scanning effects in multi-site structural mri studies. *Journal of neuroscience methods*, 230:37–50, 2014.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[CLW67] James W Cooley, Peter AW Lewis, and Peter D Welch. Historical notes on the fast fourier transform. *Proceedings of the IEEE*, 55(10):1675–1677, 1967.

[CM14] John Cheng and Ty McKercher. *Professional CUDA C programming*. John Wiley and Sons, Indianapolis, IN, 2014.

[CMPA14] Vince D Calhoun, Robyn Miller, Godfrey Pearlson, and Tulay Adalı. The chronnectome: time-varying connectivity networks as the next frontier in fmri data discovery. *Neuron*, 84(2):262–274, 2014.

[Coo10] Stephen Coombes. Large-scale neural dynamics: simple and complex. *NeuroImage*, 52(3):731–739, 2010.

[cor21] NVIDIA corporation. Cuda c++ programming guide. Technical Report PG-02829-001-v11.3, NVIDIA, April 2021.

[CSP+22] Andrew A Chen, Dhivya Srinivasan, Raymond Pomponio, Yong Fan, Ilya M Nasrallah, Susan M Resnick, Lori L Beason-Held, Christos Davatzikos, Theodore D Satterthwaite, Dani S Bassett, et al. Harmonizing functional connectivity reduces scanner effects in community detection. *NeuroImage*, 256:119198, 2022.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[CVZ+18] Sofia Chavez, Joseph Viviano, Mojdeh Zamyadi, Peter B Kingsley, Peter Kochunov, Stephen Strother, and Aristotle Voineskos. A novel dti-qa tool: Automated metric extraction exploiting the sphericity of an agar filled phantom. *Magnetic resonance imaging*, 46:28–39, 2018.

[CWP+17] Rastko Ciric, Daniel H Wolf, Jonathan D Power, David R Roalf, Graham L Baum, Kosha Ruparel, Russell T Shinohara, Mark A Elliott, Simon B Eickhoff, Christos Davatzikos, et al. Benchmarking of participant-level confound regression strategies for the control of motion artifact in studies of functional connectivity. *Neuroimage*, 154:174–187, 2017.

[DA05] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems.* MIT press, 2005.

[DBR$^+$17] Christian Dansereau, Yassine Benhajali, Celine Risterucci, Emilio Merlo Pich, Pierre Orban, Douglas Arnold, and Pierre Bellec. Statistical power and prediction accuracy in multisite resting-state fmri connectivity. *Neuroimage*, 149:220–232, 2017.

[DF13] Yuhui Du and Yong Fan. Group information guided ica for fmri data analysis. *NeuroImage*, 69:157–197, 2013.

[DFS$^+$20] Yuhui Du, Zening Fu, Jing Sui, Shuang Gao, Ying Xing, Dongdong Lin, Mustafa Salman, Anees Abrol, Md Abdur Rahaman, Jiayu Chen, et al. Neuromark: An automated and adaptive ica based pipeline to identify reproducible fmri markers of brain disorders. *NeuroImage: Clinical*, 28:102375, 2020.

[DH11] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[DL42] Gordon Charles Danielson and Cornelius Lanczos. Some improvements in practical fourier analysis and their application to x-ray scattering from liquids. *Journal of the Franklin Institute*, 233(5):435–452, 1942.

[DMOC$^+$17] Adriana Di Martino, David O'connor, Bosi Chen, Kaat Alaerts, Jeffrey S Anderson, Michal Assaf, Joshua H Balsters, Leslie Baxter, Anita Beggiato, Sylvie Bernaerts, et al. Enhancing studies of the connectome in autism using the autism brain imaging data exchange ii. *Scientific data*, 4(1):1–15, 2017.

[DMYL$^+$14] Adriana Di Martino, Chao-Gan Yan, Qingyang Li, Erin Denio, Francisco X Castellanos, Kaat Alaerts, Jeffrey S Anderson, Michal Assaf, Susan Y Bookheimer, Mirella Dapretto, et al. The autism brain imaging data exchange: towards a large-scale evaluation of the intrinsic brain architecture in autism. *Molecular psychiatry*, 19(6):659–667, 2014.

[DS00] Jack Dongarra and Francis Sullivan. Guest editors introduction to the top 10 algorithms. *Computing in Science & Engineering*, 2(01):22–23, 2000.

[DSMI11] Juergen Dukart, Matthias L Schroeter, Karsten Mueller, and Alzheimer's Disease Neuroimaging Initiative. Age correction in dementia–matching to a healthy brain. *PloS one*, 6(7):e22193, 2011.

[DZL$^+$21] Mengjiao Du, Li Zhang, Linling Li, Erni Ji, Xue Han, Gan Huang, Zhen Liang, Li Shi, Haichen Yang, and Zhiguo Zhang. Abnormal transitions of dynamic functional connectivity states in bipolar disorder: A whole-brain resting-state fmri study. *Journal of affective disorders*, 2021.

[EJCB12] Alan C Evans, Andrew L Janke, D Louis Collins, and Sylvain Baillet. Brain templates and atlases. *Neuroimage*, 62(2):911–922, 2012.

[EMF$^+$19] Taban Eslami, Vahid Mirjalili, Alvis Fong, Angela R. Laird, and Fahad Saeed. Asd-diagnet: A hybrid learning approach for detection of autism spectrum disorder using fmri data. *Frontiers in Neuroinformatics*, 13:70, 2019.

[ESM$^+$05] Simon B Eickhoff, Klaas E Stephan, Hartmut Mohlberg, Christian Grefkes, Gereon R Fink, Katrin Amunts, and Karl Zilles. A new spm toolbox for combining probabilistic cytoarchitectonic maps and functional imaging data. *Neuroimage*, 25(4):1325–1335, 2005.

[FBS22] Joshua Faskowitz, Richard F Betzel, and Olaf Sporns. Edges in brain networks: Contributions to models of structure and function. *Network Neuroscience*, 6(1):1–28, 2022.

[FCS$^+$18] Jean-Philippe Fortin, Nicholas Cullen, Yvette I Sheline, Warren D Taylor, Irem Aselcioglu, Philip A Cook, Phil Adams, Crystal Cooper, Maurizio Fava, Patrick J McGrath, et al. Harmonization of cortical thickness measurements across scanners and sites. *Neuroimage*, 167:104–120, 2018.

[FD10] Karl J Friston and Raymond J Dolan. Computational and dynamic models in neuroimaging. *Neuroimage*, 52(3):752–765, 2010.

[FEJ$^+$20] Joshua Faskowitz, Farnaz Zamani Esfahlani, Youngheun Jo, Olaf Sporns, and Richard F. Betzel. Edge-centric functional network representations of human cerebral cortex reveal overlapping system-level architecture. *Nature Neuroscience*, 23(12):1644–1654, 2020.

[FFF93] KJ Friston, CD Frith, and RSJ Frackowiak. Time-dependent changes in effective connectivity measured with pet. *Human Brain Mapping*, 1(1):69–79, 1993.

[FFLF93] Karl J Friston, Chris D Frith, Peter F Liddle, and Richard SJ Frackowiak. Functional connectivity: the principal-component analysis of large (pet) data sets. *Journal of Cerebral Blood Flow & Metabolism*, 13(1):5–14, 1993.

[FGC$^+$06] Lee Friedman, Gary H Glover, Fbirn Consortium, et al. Reducing interscanner variability of activation in a multicenter fmri study: controlling for signal-to-fluctuation-noise-ratio (sfnr) differences. *Neuroimage*, 33(2):471–481, 2006.

[Fis15] Ronald A Fisher. Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika*, 10(4):507–521, 1915.

[FIT$^+$21] Zening Fu, Armin Iraji, Jessica A Turner, Jing Sui, Robyn Miller, Godfrey D Pearlson, and Vince D Calhoun. Dynamic state with covarying brain activity-connectivity: On the pathophysiology of schizophrenia. *Neuroimage*, 224:117385, 2021.

[FMG$^+$14] Jennifer K Forsyth, Sarah C McEwen, Dylan G Gee, Carrie E Bearden, Jean Addington, Brad Goodyear, Kristin S Cadenhead, Heline Mirzakhanian, Barbara A Cornblatt, Doreen M Olvet, et al. Reliability of functional magnetic resonance imaging activation during working memory in a multi-site study: analysis from the north american prodrome longitudinal study. *Neuroimage*, 97:41–52, 2014.

[FPT$^+$17] Jean-Philippe Fortin, Drew Parker, Birkan Tunç, Takanori Watanabe, Mark A Elliott, Kosha Ruparel, David R Roalf, Theodore D Satterthwaite, Ruben C Gur, Raquel E Gur, et al. Harmonization of multi-site diffusion tensor imaging data. *Neuroimage*, 161:149–170, 2017.

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, March 1977.

[Fri07] KJ Friston. Statistical parametric mapping: the analysis of funtional brain images. *EBSCO eBooks*, 2007.

[Fri11] Karl J Friston. Functional and effective connectivity: a review. *Brain connectivity*, 1(1):13–36, 2011.

[FSB+08] Lee Friedman, Hal Stern, Gregory G Brown, Daniel H Mathalon, Jessica Turner, Gary H Glover, Randy L Gollub, John Lauriello, Kelvin O Lim, Tyrone Cannon, et al. Test–retest and between-site reliability in a multicenter fmri study. *Human brain mapping*, 29(8):958–972, 2008.

[FSF+15] Rogier A Feis, Stephen M Smith, Nicola Filippini, Gwenaëlle Douaud, Elise GP Dopper, Verena Heise, Aaron J Trachtenberg, John C van Swieten, Mark A van Buchem, Serge ARB Rombouts, et al. Ica-based artifact removal diminishes scan site differences in multi-center resting-state fmri. *Frontiers in neuroscience*, 9:395, 2015.

[FTD+19] Zening Fu, Yiheng Tu, Xin Di, Yuhui Du, Jing Sui, Bharat B Biswal, Zhiguo Zhang, Nina de Lacy, and Vincent D Calhoun. Transient increased thalamic-sensory connectivity and decreased whole-brain dynamism in autism. *Neuroimage*, 190:191–204, 2019.

[FZB15] Alex Fornito, Andrew Zalesky, and Michael Breakspear. The connectomics of brain disorders. *Nature Reviews Neuroscience*, 16(3):159–172, 2015.

[FZB16] Alex Fornito, Andrew Zalesky, and Edward Bullmore. *Fundamentals of brain network analysis.* Academic Press, Cambridge, MA 02139, USA, 2016.

[GB14] Leonardo L Gollo and Michael Breakspear. The frustrated brain: from dynamics on motifs to communities and networks. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1653):20130532, 2014.

[GCK+20] George Gifford, Nicolas Crossley, Matthew J Kempton, Sarah Morgan, Paola Dazzan, Jonathan Young, and Philip McGuire. Resting state fmri based multilayer network configuration in patients with schizophrenia. *NeuroImage: Clinical*, 25:102169, 2020.

[GDM+17] Xinyu Guo, Kelli C Dominick, Ali A Minai, Hailong Li, Craig A Erickson, and Long J Lu. Diagnosing autism spectrum disorder from brain resting-state functional connectivity patterns using a deep neural network with a novel feature selection method. *Frontiers in neuroscience*, 11:460, 2017.

[GGI+02] Anna C Gilbert, Sudipto Guha, Piotr Indyk, Shanmugavelayutham Muthukrishnan, and Martin Strauss. Near-optimal sparse fourier representations via sampling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 152–161, 2002.

[GGW+10] Victoria Gradin, Viktoria-Eleni Gountouna, Gordon Waiter, Trevor S Ahearn, David Brennan, Barrie Condon, Ian Marshall, David J McGonigle, Alison D Murray, Heather Whalley, et al. Between-and within-scanner variability in the calibrain study n-back cognitive task. *Psychiatry Research: Neuroimaging*, 184(2):86–95, 2010.

[GI10] Anna Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

[GJM+10] Viktoria-Eleni Gountouna, Dominic E Job, Andrew M McIntosh, T William J Moorhead, G Katherine L Lymer, Heather C Whalley, Jeremy Hall, Gordon D Waiter, David Brennan, David J McGonigle, et al. Functional magnetic resonance imaging (fmri) reproducibility and variance components across visits and scanning sites with a finger tapping task. *Neuroimage*, 49(1):552–560, 2010.

[GMS05] Anna C Gilbert, Shan Muthukrishnan, and Martin Strauss. Improved time bounds for near-optimal sparse fourier representations. In *Wavelets XI*, volume 5914, pages 398–412. SPIE, 2005.

[GMT+12] Gary H Glover, Bryon A Mueller, Jessica A Turner, Theo GM Van Erp, Thomas T Liu, Douglas N Greve, James T Voyvodic, Jerod Rasmussen, Gregory G Brown, David B Keator, et al. Function biomedical informatics research network recommendations for prospective multicenter functional mri studies. *Journal of Magnetic Resonance Imaging*, 36(1):39–54, 2012.

[GOB+10] Nils Gehlenborg, Seán I O'donoghue, Nitin S Baliga, Alexander Goesmann, Matthew A Hibbs, Hiroaki Kitano, Oliver Kohlbacher, Heiko Neuweger, Reinhard Schneider, Dan Tenenbaum, et al. Visualization of omics data for systems biology. *Nature methods*, 7(Suppl 3):S56–S68, 2010.

[GSC$^+$22] Yanyan Gao, Jiawei Sun, Lulu Cheng, Qihang Yang, Jing Li, Zeqi Hao, Linlin Zhan, Yuyu Shi, Mengting Li, Xize Jia, et al. Altered resting state dynamic functional connectivity of amygdala subregions in patients with autism spectrum disorder: A multi-site fmri study. *Journal of Affective Disorders*, 2022.

[Gui20] Design Guide. Cuda c++ best practices guide. 2020.

[Hag05] Patric Hagmann. From diffusion mri to brain connectomics. Technical report, EPFL, 2005.

[HAM$^+$16] Rikkert Hindriks, Mohit H Adhikari, Yusuke Murayama, Marco Ganzetti, Dante Mantini, Nikos K Logothetis, and Gustavo Deco. Can sliding-window correlations reveal dynamic functional connectivity in resting-state fmri? *Neuroimage*, 127:242–256, 2016.

[Has18] Haitham Hassanieh. *The Sparse Fourier Transform*. Morgan & Claypool, 2018.

[HB18] Stewart Heitmann and Michael Breakspear. Putting the "dynamic" back into dynamic functional connectivity. *Network Neuroscience*, 2(02):150–174, 2018.

[HFC$^+$18] Anibal Sólon Heinsfeld, Alexandre Rosa Franco, R Cameron Craddock, Augusto Buchweitz, and Felipe Meneguzzi. Identification of autism spectrum disorder using deep learning and the abide dataset. *NeuroImage: Clinical*, 17:16–23, 2018.

[HIKP12] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Sparse fast fourier transform code documentation (sfft 1.0 and 2.0). In *STOC*, 2012.

[HRGCB12] Daniel A Handwerker, Vinai Roopchansingh, Javier Gonzalez-Castillo, and Peter A Bandettini. Periodic changes in fmri connectivity. *Neuroimage*, 63(3):1712–1719, 2012.

[HWA$^+$13] R Matthew Hutchison, Thilo Womelsdorf, Elena A Allen, Peter A Bandettini, Vince D Calhoun, Maurizio Corbetta, Stefania Della Penna, Jeff H Duyn, Gary H Glover, Javier Gonzalez-Castillo, et al. Dynamic functional connectivity: promise, issues, and interpretations. *Neuroimage*, 80:360–378, 2013.

[HWG$^+$13] R Matthew Hutchison, Thilo Womelsdorf, Joseph S Gati, Stefan Everling, and Ravi S Menon. Resting-state networks show dynamic functional connectivity in awake humans and anesthetized macaques. *Human brain mapping*, 34(9):2154–2177, 2013.

[IAV$^+$23] Sebastian Idesis, Michele Allegra, Jakub Vohryzek, Yonatan Sanz Perl, Joshua Faskowitz, Olaf Sporns, Maurizio Corbetta, and Gustavo Deco. A low dimensional embedding of brain dynamics enhances diagnostic accuracy and behavioral prediction in stroke. *Scientific Reports*, 13(1):15698, 2023.

[Iid15] Tetsuya Iidaka. Resting state functional magnetic resonance imaging and neural network classified autism and control. *Cortex*, 63:55–67, 2015.

[JBB13] Heidi Johansen-Berg and Timothy EJ Behrens. *Diffusion MRI: from quantitative measurement to in vivo neuroanatomy.* Academic Press, 2013.

[JC18] Mark Jenkinson and Michael Chappell. *Introduction to neuroimaging analysis.* Oxford University Press, 2018.

[JLH$^+$12] Yuntao Jia, Victor Lu, Jared Hoberock, Michael Garland, and John C. Hart. Chapter 2 - edge v. node parallelism for graph centrality metrics. In *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 15–28. Morgan Kaufmann, Boston, 2012.

[JLR07] W Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127, 2007.

[JMS$^+$01] Peter Jezzard, Paul M Matthews, Stephen M Smith, et al. *Functional MRI: an introduction to methods*, volume 61. Oxford university press Oxford, 2001.

[JVM$^+$12] David T Jones, Prashanthi Vemuri, Matthew C Murphy, Jeffrey L Gunter, Matthew L Senjem, Mary M Machulda, Scott A Przybelski, Brian E Gregg, Kejal Kantarci, David S Knopman, et al. Non-stationarity in the “resting brain’s” modular architecture. *PloS one*, 7(6):e39731, 2012.

[JWBW16] Wenjun Jiang, Guojun Wang, Md Zakirul Alam Bhuiyan, and Jie Wu. Understanding graph-based trust evaluation in online social networks: Methodologies and challenges. *Acm Computing Surveys (Csur)*, 49(1):1–35, 2016.

[KAA$^+$18] Robert E Kass, Shun-Ichi Amari, Kensuke Arai, Emery N Brown, Casey O Diekman, Markus Diesmann, Brent Doiron, Uri T Eden, Adrienne L Fairhall, Grant M Fiddyment, et al. Computational neuroscience: Mathematical and statistical perspectives. *Annual review of statistics and its application*, 5:183–214, 2018.

[KAB$^+$19] Scott P. Kolodziej, Mohsen Aznaveh, Matthew Bullock, Jarrett David, Timothy A. Davis, Matthew Henderson, Yifan Hu, and Read Sandstrom. The suitesparse matrix collection website interface. *Journal of Open Source Software*, 4(35):1244, 2019.

[KAD$^+$14] Daniel Kostro, Ahmed Abdulkadir, Alexandra Durr, Raymund Roos, Blair R Leavitt, Hans Johnson, David Cash, Sarah J Tabrizi, Rachael I Scahill, Olaf Ronneberger, et al. Correction of inter-scanner and within-subject variance in structural mri based automated diagnosing. *NeuroImage*, 98:405–415, 2014.

[KD18] Nikolaus Kriegeskorte and Pamela K Douglas. Cognitive computational neuroscience. *Nature neuroscience*, 21(9):1148–1160, 2018.

[KD20] Morten L Kringelbach and Gustavo Deco. Brain states and transitions: insights from computational neuroscience. *Cell Reports*, 32(10), 2020.

[KEFK$^+$17] Aya Kabbara, W El Falou, M Khalil, F Wendling, and M Hassan. The dynamic functional core network of the human brain at rest. *Scientific reports*, 7(1):2936, 2017.

[KFMB$^+$16] Pegah Kassraian-Fard, Caroline Matthis, Joshua H Balsters, Marloes H Maathuis, and Nicole Wenderoth. Promises, pitfalls, and basic guidelines for applying machine learning classifiers to psychiatric imaging data, with autism as an example. *Frontiers in psychiatry*, 7:177, 2016.

[KG11] Jeremy Kepner and John Gilbert. *Graph Algorithms in the Language of Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.

[KGX+19] Yazhou Kong, Jianliang Gao, Yunpei Xu, Yi Pan, Jianxin Wang, and Jin Liu. Classification of autism spectrum disorder by combining brain connectivity and deep neural network classifier. *Neurocomputing*, 324:63–68, 2019.

[KJKS19] Meenakshi Khosla, Keith Jamison, Amy Kuceyeski, and Mert R Sabuncu. Ensemble learning with 3d convolutional neural networks for functional connectome-based prediction. *NeuroImage*, 199:651–662, 2019.

[KSBB18] Ankit N Khambhati, Ann E Sizemore, Richard F Betzel, and Danielle S Bassett. Modeling and interpreting mesoscale network dynamics. *NeuroImage*, 180:337–349, 2018.

[KSL17] Tae-Eui Kam, Heung-Il Suk, and Seong-Whan Lee. Multiple functional networks modeling for autism spectrum disorder diagnosis. *Human brain mapping*, 38(11):5804–5821, 2017.

[KWM16] David B Kirk and W Hwu Wen-Mei. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[LAT+05] Lun Li, David Alderson, Reiko Tanaka, John C. Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications (extended version). Technical Report CIT-CDS-04-006, California Institute of Technology, 2005.

[LGD+20] Xiaoxiao Li, Yufeng Gu, Nicha Dvornek, Lawrence H Staib, Pamela Ventola, and James S Duncan. Multi-site fmri analysis using privacy-preserving federated learning and domain adaptation: Abide results. *Medical Image Analysis*, 65:101765, 2020.

[LK14] Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection, 2014.

[LKB+20] Daniel J Lurie, Daniel Kessler, Danielle S Bassett, Richard F Betzel, Michael Breakspear, Shella Kheilholz, Aaron Kucyi, Raphaël Liégeois, Martin A Lindquist, Anthony Randal McIntosh, et al. Questions and controversies in the study of time-varying functional connectivity in resting fmri. *Network Neuroscience*, 4(1):30–69, 2020.

[LNF17] Thomas T Liu, Alican Nalci, and Maryam Falahpour. The global signal in fmri: Nuisance or information? *Neuroimage*, 150:213–229, 2017.

[LS16] Jure Leskovec and Rok Sosic. Snap: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, 8(1):1–20, 2016.

[LSM$^+$17] Timothy O Laumann, Abraham Z Snyder, Anish Mitra, Evan M Gordon, Caterina Gratton, Babatunde Adeyemo, Adrian W Gilmore, Steven M Nelson, Jeff J Berg, Deanna J Greene, et al. On the stability of bold fmri correlations. *Cerebral cortex*, 27(10):4719–4732, 2017.

[LVDV15] Nora Leonardi and Dimitri Van De Ville. On spurious and real fluctuations of dynamic functional connectivity during rest. *Neuroimage*, 104:430–436, 2015.

[MAS$^+$19] Fatemeh Mokhtari, Milad I Akhlaghi, Sean L Simpson, Guorong Wu, and Paul J Laurienti. Sliding window correlation analysis: Modulating window shape for dynamic brain connectivity in resting state. *Neuroimage*, 189:655–666, 2019.

[MB14] Adam McLaughlin and David A. Bader. Scalable and high performance betweenness centrality on the gpu. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 572–583, New Orleans, Louisana, 2014. IEEE Press.

[Mes20] Arnaud Messé. Parcellation influence on the connectivity-based structure–function relationship in the human brain. *Human Brain Mapping*, 41(5):1167–1180, 2020.

[MF17] Kevin Murphy and Michael D Fox. Towards a consensus regarding global signal regression for resting state functional connectivity mri. *Neuroimage*, 154:169–173, 2017.

[MMO19] Teppei Matsui, Tomonari Murakami, and Kenichi Ohki. Neuronal origin of the temporal dynamics of spontaneous bold activity correlation. *Cerebral Cortex*, 29(4):1496–1508, 2019.

[MNS$^+$16] Hengameh Mirzaalian, Lipeng Ning, Peter Savadjiev, Ofer Pasternak, Sylvain Bouix, O Michailovich, Gerald Grant, Christine E Marx, Rajendra A Morey, Laura A Flashman, et al. Inter-site and inter-scanner diffusion mri data harmonization. *NeuroImage*, 135:311–323, 2016.

[MR06] Alain J Marengo-Rowe. Structure-function relations of human hemoglobins. In *Baylor University Medical Center Proceedings*, volume 19, pages 239–245. Taylor & Francis, 2006.

[MWBV18] Sarah E. Morgan, Simon R. White, Edward T. Bullmore, and Petra E. Vértes. A network neuroscience approach to typical and atypical brain development. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, 3(9):754–766, 2018. Computational Methods and Modeling in Psychiatry.

[Ney92] Jerzy Neyman. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. In *Breakthroughs in statistics*, pages 123–150. Springer, 1992.

[NSF+17] Stephanie Noble, Dustin Scheinost, Emily S Finn, Xilin Shen, Xenophon Papademetris, Sarah C McEwen, Carrie E Bearden, Jean Addington, Bradley Goodyear, Kristin S Cadenhead, et al. Multisite reliability of mr-based functional connectivity. *Neuroimage*, 146:959–970, 2017.

[NVD+17] Jason S Nomi, Shruti Gopal Vij, Dina R Dajani, Rosa Steimke, Eswar Damaraju, Srinivas Rachakonda, Vince D Calhoun, and Lucina Q Uddin. Chronnectomic patterns and neural flexibility underlie executive function. *Neuroimage*, 147:861–871, 2017.

[NVI] CUDA NVIDIA. Programming guide, cusparse, cublas, and cufft library user guides.

[NZF+13] Jared A Nielsen, Brandon A Zielinski, P Thomas Fletcher, Andrew L Alexander, Nicholas Lange, Erin D Bigler, Janet E Lainhart, and Jeffrey S Anderson. Multisite functional connectivity mri classification of autism: Abide results. *Frontiers in human neuroscience*, 7:599, 2013.

[OAS10] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245–251, 2010.

[OLNG90] Seiji Ogawa, Tso-Ming Lee, Asha S Nayak, and Paul Glynn. Oxygenation-sensitive contrast in magnetic resonance image of rodent brain at high magnetic fields. *Magnetic resonance in medicine*, 14(1):68–78, 1990.

[OMT+93] Seiji Ogawa, RS Menon, David W Tank, SG Kim, H Merkle, JM Ellermann, and K Ugurbil. Functional brain mapping by blood oxygenation level-dependent contrast magnetic resonance imaging. a comparison of signal characteristics with a biophysical model. *Biophysical journal*, 64(3):803–812, 1993.

[Par14] Van L Parsons. Stratified sampling. *Wiley StatsRef: Statistics Reference Online*, pages 1–11, 2014.

[PBG+12] Jean-Baptiste Poline, Janis L Breeze, Satrajit Ghosh, Krzysztof Gorgolewski, Yaroslav O Halchenko, Michael Hanke, Christian Haselgrove, Karl G Helmer, David B Keator, Daniel S Marcus, et al. Data sharing in neuroimaging research. *Frontiers in neuroinformatics*, 6:9, 2012.

[PBM15] Mark Plitt, Kelly Anne Barnes, and Alex Martin. Functional connectivity classification of autism identifies highly predictive brain features but falls short of biomarker standards. *NeuroImage: Clinical*, 7:359–366, 2015.

[PBVDV17] Maria Giulia Preti, Thomas AW Bolton, and Dimitri Van De Ville. The dynamic functional connectome: State-of-the-art and perspectives. *Neuroimage*, 160:41–54, 2017.

[PF13] Hae-Jeong Park and Karl Friston. Structural and functional brain networks: from connections to cognition. *Science*, 342(6158), 2013.

[PKF+18] Sarah Parisot, Sofia Ira Ktena, Enzo Ferrante, Matthew Lee, Ricardo Guerrero, Ben Glocker, and Daniel Rueckert. Disease prediction using graph convolutional networks: application to autism spectrum disorder and alzheimer's disease. *Medical image analysis*, 48:117–130, 2018.

[PSP15] Jonathan D Power, Bradley L Schlaggar, and Steven E Petersen. Recent progress and outstanding issues in motion correction in resting state fmri. *Neuroimage*, 105:536–551, 2015.

[PWW$^+$17] Yuechao Pan, Yangzihao Wang, Yuduo Wu, Carl Yang, and J. D. Owens. Multi-gpu graph analytics. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 479–490, Orlando, Florida, 2017. IEEE Press.

[RAD$^+$16] Barnaly Rashid, Mohammad R Arbabshirani, Eswar Damaraju, Mustafa S Cetin, Robyn Miller, Godfrey D Pearlson, and Vince D Calhoun. Classification of schizophrenia and bipolar patients using static and dynamic resting-state fmri brain connectivity. *Neuroimage*, 134:645–657, 2016.

[RJF$^+$21] Maya A Reiter, Afrooz Jahedi, AR Fredo, Inna Fishman, Barbara Bailey, and Ralph-Axel Müller. Performance of machine learning classification models of autism using resting-state fmri is contingent on sample heterogeneity. *Neural Computing and Applications*, 33(8):3299–3310, 2021.

[RLH21] Alexandra M Reardon, Kaiming Li, and Xiaoping P Hu. Improving between-group effect size for multi-site functional connectivity data via site-wise de-meaning. *Frontiers in computational neuroscience*, 15(762781):111, 2021.

[RMMM$^+$17] Anil Rao, Joao M Monteiro, Janaina Mourao-Miranda, Alzheimer's Disease Initiative, et al. Predictive modelling using neuroimaging data in the presence of confounds. *NeuroImage*, 150:23–49, 2017.

[RS10] Mikail Rubinov and Olaf Sporns. Complex network measures of brain connectivity: uses and interpretations. *Neuroimage*, 52(3):1059–1069, 2010.

[RyC06] Santiago Ramón y Cajal. The structure and connexions of neurons. *Nobel Lecture*, 12, 1906.

[SAS$^+$20] Zeinab Sherkatghanad, Mohammadsadegh Akhondzadeh, Soorena Salari, Mariam Zomorodi-Moghadam, Moloud Abdar, U Rajendra Acharya, Reza Khosrowabadi, and Vahid Salari. Automated detection of autism spectrum disorder using a convolutional neural network. *Frontiers in neuroscience*, 13:1325, 2020.

[SB13] Julian Shun and Guy E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. In *Proceedings of the 18th*

ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 135–146, Shenzhen, China, 2013. ACM.

[SCKH04] Olaf Sporns, Dante R Chialvo, Marcus Kaiser, and Claus C Hilgetag. Organization, development and function of complex brain networks. *Trends in cognitive sciences*, 8(9):418–425, 2004.

[SHK07] Olaf Sporns, Christopher J Honey, and Rolf Kötter. Identification and classification of hubs in brain networks. *PloS one*, 2(10):e1049, 2007.

[Shl14] Jonathon Shlens. A tutorial on independent component analysis. *arXiv preprint arXiv:1404.2986*, 2014.

[SI06] JO Smith III. Mathematics of the discrete fourier transform with audio applications. *Booksurge LLC*, 2006.

[SJD$^+$23] Caio Seguin, Maciej Jedynak, Olivier David, Sina Mansour, Olaf Sporns, and Andrew Zalesky. Communication dynamics in the human connectome shape the cortex-wide propagation of direct electrical stimulation. *Neuron*, 111(9):1391–1401, 2023.

[SKB$^+$17] Masoumeh Sadeghi, Reza Khosrowabadi, Fatemeh Bakouie, Hoda Mahdavi, Changiz Eslahchi, and Hamidreza Pouretemad. Screening of autism based on task-free fmri using graph theoretical approach. *Psychiatry Research: Neuroimaging*, 263:48–56, 2017.

[SKSC13] Ahmet Erdem Sariyuce, Kamer Kaya, Erik Saule, and Umit V. Catalyurek. Betweenness centrality on gpus and heterogeneous architectures. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, pages 76–85, Houston, Texas, USA, 2013. ACM.

[SLK16] Sadia Shakil, Chin-Hui Lee, and Shella Dawn Keilholz. Evaluation of sliding window correlation performance for characterizing dynamic functional connectivity and brain states. *Neuroimage*, 133:111–128, 2016.

[SMD04] Felice T Sun, Lee M Miller, and Mark D'esposito. Measuring interregional functional connectivity using coherence and partial coherence analyses of fmri data. *Neuroimage*, 21(2):647–658, 2004.

[SMM$^+$16] José M Soares, Ricardo Magalhães, Pedro S Moreira, Alexandre Sousa, Edward Ganz, Adriana Sampaio, Victor Alves, Paulo Marques, and Nuno Sousa. A hitchhiker's guide to functional magnetic resonance imaging. *Frontiers in neuroscience*, 10:515, 2016.

[SON$^+$17] Russell T Shinohara, Jiwon Oh, Govind Nair, Peter A Calabresi, Christos Davatzikos, Jimit Doshi, Roland G Henry, Gloria Kim, Kristin A Linn, Nico Papinutto, et al. Volumetric analysis from a harmonized multisite brain mri study of a single subject with multiple sclerosis. *American Journal of Neuroradiology*, 38(8):1501–1509, 2017.

[SP14] Jörn Schumacher and Markus Püschel. High-performance sparse fast fourier transforms. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2014.

[SPK$^+$10] Ünal Sakoğlu, Godfrey D Pearlson, Kent A Kiehl, Y Michelle Wang, Andrew M Michael, and Vince D Calhoun. A method for evaluating dynamic functional network connectivity and task-modulation: application to schizophrenia. *Magnetic Resonance Materials in Physics, Biology and Medicine*, 23(5):351–366, 2010.

[Spo11] Olaf Sporns. The human connectome: a complex network. *Annals of the new York Academy of Sciences*, 1224(1):109–125, 2011.

[Spo12] Olaf Sporns. From simple graphs to the connectome: Networks in neuroimaging. *NeuroImage*, 62(2):881–886, 2012. 20 YEARS OF fMRI.

[Spo13] Olaf Sporns. Structure and function of complex brain networks. *Dialogues in clinical neuroscience*, 15(3):247, 2013.

[Spo18] Olaf Sporns. Graph theory methods: applications in brain networks. *Dialogues in clinical neuroscience*, 20(2):111, 2018.

[SR07] Cornelis J. Stam and Jaap C. Reijneveld. Graph theoretical analysis of complex networks in the brain. *Nonlinear Biomedical Physics*, 1(1):3, 2007.

[SS16] Peter Sprent and Nigel C Smeeton. *Applied nonparametric statistical methods*. CRC press, Boca Raton, FL, 33487-2742, 2016.

[SS20] Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524, 2020.

[SSZ23] Caio Seguin, Olaf Sporns, and Andrew Zalesky. Brain network communication: concepts, models and applications. *Nature Reviews Neuroscience*, pages 1–18, 2023.

[STK05] Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: a structural description of the human brain. *PLoS computational biology*, 1(4):e42, 2005.

[SZ11] Zhiao Shi and Bing Zhang. Fast network centrality analysis using gpus. *BMC Bioinformatics*, 12(1):149, 2011.

[TD00] Ajit Tamhane and Dunlop Dunlop. *Statistics and data analysis: from elementary to intermediate*. Prentice-Hall, Upper Sadle River, NJ 07458, 2000.

[TEL$^+$92] James Theiler, Stephen Eubank, André Longtin, Bryan Galdrikian, and J Doyne Farmer. Testing for nonlinearity in time series: the method of surrogate data. *Physica D: Nonlinear Phenomena*, 58(1-4):77–94, 1992.

[TMA$^+$21] Mahbaneh Eshaghzadeh Torbati, Davneet S Minhas, Ghasan Ahmad, Erin E O'Connor, John Muschelli, Charles M Laymon, Zixi Yang, Ann D Cohen, Howard J Aizenstein, William E Klunk, et al. A multi-scanner neuroimaging data harmonization using ravel and combat. *NeuroImage*, 245:118703, 2021.

[TMLP$^+$02] Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Olivier Etard, Nicolas Delcroix, Bernard Mazoyer, and Marc Joliot. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain. *Neuroimage*, 15(1):273–289, 2002.

[TVWM$^+$12] Enzo Tagliazucchi, Frederic Von Wegner, Astrid Morzelewski, Verena Brodbeck, and Helmut Laufs. Dynamic bold functional connectivity in humans and its electrophysiological correlates. *Frontiers in human neuroscience*, 6:339, 2012.

[vdHSBP08] Martijn P van den Heuvel, Cornelis J Stam, Maria Boersma, and HE Hulshoff Pol. Small-world and scale-free organization of voxel-based resting-state functional connectivity in the human brain. *Neuroimage*, 43(3):528–539, 2008.

[vdVFP$^+$04] Vincent G van de Ven, Elia Formisano, David Prvulovic, Christian H Roeder, and David EJ Linden. Functional connectivity as revealed by spatial independent component analysis of fmri measurements during rest. *Human brain mapping*, 22(3):165–178, 2004.

[VG20] Todd Vanderah and Douglas Gould. *Nolte's The Human Brain E-Book: An Introduction to its Functional Anatomy.* Elsevier Health Sciences, 2020.

[VHT09] John Darrell Van Horn and Arthur W Toga. Multisite neuroimaging trials. *Current opinion in neurology*, 22(4):370, 2009.

[VMSS13] S Vigneshwaran, BS Mahanand, Sundaram Suresh, and Ramaswamy Savitha. Autism spectrum disorder detection using projection based learning meta-cognitive rbf network. In IEEE, editor, *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, New York, USA, 2013. IEEE, IEEE Press.

[VS13] Tyler J VanderWeele and Ilya Shpitser. On the definition of a confounder. *Annals of statistics*, 41(1):196, 2013.

[WBQ$^+$14] Huifang E Wang, Christian G Bénar, Pascale P Quilichini, Karl J Friston, Viktor K Jirsa, and Christophe Bernard. A systematic framework for functional connectivity measures. *Frontiers in neuroscience*, 8:405, 2014.

[WCC16] Cheng Wang, Sunita Chandrasekaran, and Barbara Chapman. cusfft: A high-performance sparse fast fourier transform algorithm on gpus. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 963–972. IEEE, 2016.

[WDP$^+$16] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D. Owens. Gunrock: A high-performance graph processing library on the gpu. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, Barcelona, Spain, 2016. ACM.

[WXW19] Canhua Wang, Zhiyong Xiao, and Jianhua Wu. Functional connectivity-based classification of autism and control using svm-rfecv on rs-fmri data. *Physica Medica*, 65:99–105, 2019.

[XSQ+18] Huaze Xu, Jianpo Su, Jian Qin, Ming Li, Ling-Li Zeng, Dewen Hu, and Hui Shen. Impact of global signal regression on characterizing dynamic functional connectivity and brain states. *Neuroimage*, 173:127–145, 2018.

[YBO22] Carl Yang, Aydın Buluç, and John D Owens. Graphblast: A high-performance linear algebra-based graph framework on the gpu. *ACM Transactions on Mathematical Software (TOMS)*, 48(1):1–51, 2022.

[yC97] Santiago Ramón y Cajal. *Leyes de la morfología y dinamismo de las células nerviosas.* N. Moya, 1897.

[YHX+15] Zhijun Yao, Bin Hu, Yuanwei Xie, Philip Moore, and Jiaxiang Zheng. A review of structural and functional brain networks: small world and atlas. *Brain informatics*, 2(1):45–52, 2015.

[YKK17] Noriaki Yahata, Kiyoto Kasai, and Mitsuo Kawato. Computational neuroscience approach to biomarkers and treatments for mental disorders. *Psychiatry and clinical neurosciences*, 71(4):215–237, 2017.

[YLC+18] Meichen Yu, Kristin A Linn, Philip A Cook, Mary L Phillips, Melvin McInnis, Maurizio Fava, Madhukar H Trivedi, Myrna M Weissman, Russell T Shinohara, and Yvette I Sheline. Statistical harmonization corrects site effects in functional connectivity measurements from multi-site fmri data. *Human brain mapping*, 39(11):4213–4227, 2018.

[YYI+19] Ayumu Yamashita, Noriaki Yahata, Takashi Itahashi, Giuseppe Lisi, Takashi Yamada, Naho Ichikawa, Masahiro Takamura, Yujiro Yoshihara, Akira Kunimatsu, Naohiro Okada, et al. Harmonization of resting-state functional mri data across multiple imaging sites via the separation of site differences into sampling bias and measurement bias. *PLoS biology*, 17(4):e3000042, 2019.

[ZB15] Andrew Zalesky and Michael Breakspear. Towards a statistical test for functional connectivity dynamics. *Neuroimage*, 114:466–470, 2015.

[ZFC+14] Andrew Zalesky, Alex Fornito, Luca Cocchi, Leonardo L Gollo, and Michael Breakspear. Time-resolved resting-state brain networks. *Pro-*

ceedings of the National Academy of Sciences, 111(28):10341–10346, 2014.

[ZXS$^+$22] Lei Zhao, Shao-Wei Xue, Yun-Kai Sun, Zhihui Lan, Ziqi Zhang, Yichen Xue, Xuan Wang, and Yuxin Jin. Altered dynamic functional connectivity of insular subregions could predict symptom severity of male patients with autism spectrum disorder. *Journal of Affective Disorders*, 299:504–512, 2022.

[ZZC$^+$17] Yu Zhang, Han Zhang, Xiaobo Chen, Seong-Whan Lee, and Dinggang Shen. Hybrid high-order functional connectivity networks using resting-state functional mri for mild cognitive impairment diagnosis. *Scientific reports*, 7(1):6530, 2017.

[ZZW$^+$10] Wen Zhu, Nancy Zeng, Ning Wang, et al. Sensitivity, specificity, accuracy, associated confidence interval and roc analysis with practical sas implementations. *NESUG proceedings: health care and life sciences, Baltimore, Maryland*, 19:67, 2010.

APPENDIX A: CODE AVAILABILITY

All the software codes implemented to compute the experimental results included in this dissertation are available as open-source software as indicated in the following list.

1. GPU-SFFT, chapter 2,
   https://github.com/pcdslab/gpu-sfft, Languages: CUDA, C.
2. TurboBFS, chapter 3,
   https://github.com/pcdslab/TurboBFS, Languages: CUDA, C.
3. TurboBC, chapter 4,
   https://github.com/pcdslab/TurboBC, Languages: CUDA, C.
4. ASD-DiagNet-Confounds, chapter 5,
   https://github.com/pcdslab/ASD-DiagNet-Confounds, Language: Python.
5. Time-varying-FC, chapter 6,
   Language: Python, development stage.

APPENDIX B: DATA AVAILABILITY

The ABIDE preprocessed rs-fMRI multi-site database used in this dissertation is available at the website (http://preprocessed-connectomes-project.org/abide).

APPENDIX C: COMPUTING RESOURCES AND FUNDING

COMPUTING RESOURCES The work presented in this dissertation used the Raptor and Dragon High-Performance Computing cluster resources at the Florida International University supported by the NIH Supplemental Award: 3R01GM134384-02S1.

FUNDING The work presented in this dissertation was funded by the National Science Foundation (NSF) award No. OAC 2312599, and by grant of the National Institute of Health (NIH) No. R01GM134384. Any opinions, findings, and conclusions or recommendations expressed in the content of this dissertation are solely the responsibility of the author and do not necessarily represent the views of the National Science Foundation (NSF) and/or the National Institute of Health (NIH).

VITA

OSWALDO ARTILES

	Born, Caracas, Venezuela
1970	Electrical Engineering degree, Universidad Central de Venezuela, Caracas, Venezuela
1970-2003	Engineer and manager of EDELCA, electrical utility, owner of one of the biggest hydroelectric complex in the world, Bolivar, Venezuela
1977	MsC in Power systems enginnering, University of Manchester Institute of Science and Technology (UMIST), Manchester, England
2017	PhD, Physics, Florida International University, Miami, Florida, USA
2023	PhD, Computer Science, Florida International University, Miami, Florida, USA

PUBLICATIONS AND PRESENTATIONS

1. Artiles, Oswaldo, Zeina Al Masry, and Fahad Saeed. *Confounding Effects on the Performance of Machine Learning Analysis of Static Functional Connectivity Computed from rs-fMRI Multi-site Data.* Neuroinformatics (2023): 1-18.
2. Artiles, Oswaldo, and Fahad Saeed. *A Multi-Factorial Assessment of Functional Human Autistic Spectrum Brain Network Analysis.* 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). IEEE, 2021.
3. Artiles, Oswaldo, and Fahad Saeed. *TurboBC: A Memory Efficient and Scalable GPU Based Betweenness Centrality Algorithm in the Language of Linear Algebra.* 50th International Conference on Parallel Processing Workshop. 2021.
4. Artiles, Oswaldo, and Fahad Saeed. *TurboBFS: GPU based breadth-first search (BFS) algorithms in the language of linear algebra.* 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2021.
5. Artiles, Oswaldo, and Fahad Saeed. *GPU-SFFT: A GPU based parallel algorithm for computing the Sparse Fast Fourier Transform (SFFT) of k-sparse signals.* 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019.
6. Artiles, Oswaldo. *Multinucleon Short-range Correlation Model for Nuclear Spectral Functions.* PhD Dissertation, arXiv preprint arXiv:1805.02778 (2018).
7. Artiles, Oswaldo, and Misak M. Sargsian. *Multinucleon short-range correlation model for nuclear spectral functions: theoretical framework*, Physical Review C 94.6 (2016): 064318.