UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Dottorato in Informatica
XXXIII Ciclo

Dipartimento di Informatica "Giovanni Degli Antoni"

Tesi di Dottorato di Ricerca

High Performance Computing Machine Learning Methods For Precision Medicine

Relatori:
Prof. Giorgio Valentini
Prof. Giuliano Grossi

Candidato:
Alessandro Petrini

Coordinatore:
Prof. Paolo Boldi

Anno accademico 2019 / 2020

Abstract

Precision Medicine is a new paradigm which is reshaping several aspects of clinical practice, representing a major departure from the "one size fits all" approach in diagnosis and prevention featured in classical medicine. Its main goal is to find personalized prevention measures and treatments, on the basis of the personal history, lifestyle and specific genetic factors of each individual.

Three factors contributed to the rapid rise of Precision Medicine approaches: the ability to quickly and cheaply generate a vast amount of biological and omics data, mainly thanks to Next-Generation Sequencing; the ability to efficiently access this vast amount of data, under the Big Data paradigm; the ability to automatically extract relevant information from data, thanks to innovative and highly sophisticated data processing analytical techniques.

Machine Learning in recent years revolutionized data analysis and predictive inference, influencing almost every field of research. Moreover, high-throughput bio-technologies posed additional challenges to effectively manage and process Big Data in Medicine, requiring novel specialized Machine Learning methods and High Performance Computing techniques well-tailored to process and extract knowledge from big bio-medical data.

In this thesis we present three High Performance Computing Machine Learning techniques that have been designed and developed for tackling three fundamental and still open questions in the context of Precision and Genomic Medicine: i) identification of pathogenic and deleterious genomic variants among the "sea" of neutral variants in the non-coding regions of the DNA; ii) detection of the activity of regulatory regions across different cell lines and tissues; iii) automatic protein function prediction and drug repurposing in the context of biomolecular networks.

For the first problem we developed *parSMURF*, a novel hyper-ensemble method able to deal with the huge data imbalance that characterizes the detection of pathogenic variants in the non-coding regulatory regions of the human genome. We implemented this approach with highly parallel computational techniques using supercomputing resources at CINECA (Marconi –

KNL) and HPC Center Stuttgart (HLRS Apollo HAWK), obtaining state-of-the-art results. For the second problem we developed Deep Feed Forward and Deep Convolutional Neural Networks to respectively process epigenetic and DNA sequence data to detect active promoters and enhancers in specific tissues at genome-wide level using GPU devices to parallelize the computation. Finally we developed scalable semi-supervised graph-based Machine Learning algorithms based on parametrized Hopfield Networks to process in parallel using GPU devices large biological graphs, using a parallel coloring method that improves the classical Luby greedy algorithm.

We also present ongoing extensions of *parSMURF*, very recently awarded by the Partnership for Advance in Computing in Europe (PRACE) consortium to further develop the algorithm, apply them to huge genomic data and embed its results into Genomiser, a state-of-the-art computational tool for the detection of pathogenic variants associated with Mendelian genetic diseases, in the context of an international collaboration with the Jackson Lab for Genomic Medicine.

Table of Contents

List of abbreviations and terms

AFP	Automated Function Prediction (of proteins)
AI	Artificial Intelligence
ANN	Artificial Neural Network
AUPRC	Area Under the Precision-Recall Curve
AUROC	Area Under the Receiver-Operating Characteristic Curve
BLAST	Basic Local Alignment Search Tool
BO	Bayesian Optimization
CADD	Combined Annotation-Dependent Depletion
CAGE	Cap Analysis of Gene Expression
CNN	Convolutional Neural Network
COSNet	COst Sensitive neural Network
CPU	Central Processing Unit
CRE	Cis-Regulatory Element
CUDA	Compute Unified Device Architecture
CV	Cross-Validation
DL	Deep Learning
DNN	Deep Neural Network
ENCODE	Encyclopedia of DNA Elements
FANTOM	Functional Annotation Of the Mammalian Genome
FATHMM-MKL	Functional Analysis through Hidden Markov Models and Multiple Kernel Learning
FFNN	Feed-Forward Neural Network
G/C content	Guanine-Cytosine content
GGC	Greedy Graph Coloring
gkm-SVM	Gapped k-mer Support Vector Machine
GO	Gene Ontology
GPU	Graphics processing unit
GWAS	Genome-Wide Association Study
GWAVA	Genome Wide Annotation of Variants
HN	Hopfield Network

HPC	High Performance Computing
MCA	Multiple Correspondence Analysis
MCMC	Markov Chain Monte Carlo
MIC	Many Integrated Core Architecture
MIS	Maximal Independent Set
ML	Machine Learning
MPI	Message Passing Interface
NGS	Next Generation Sequencing
OpenMP	Open Multi-Processing
PCA	Principal Components Analysis
PM	Precision Medicine
PMD	Poisson Multinomial Distribution
PRACE	Partnership for Advanced Computing in Europe consortium
ReLU	Rectified Linear Unit
ReMM	Regulatory Mendelian Mutations
RF	Random Forest
SGD	Stochastic Gradient Descent
SIMT	Single Instruction Multiple Thread
SIMD	Single Instruction Multiple Data
SM	Streaming Multiprocessors
SMOTE	Synthetic Minority Over-Sample technique
SNP	Single Nucleotide Polymorphism
SNV	Single Nucleotide Variants
SVM	Support Vector Machine
t-SNE	T-distributed stochastic neighbor embedding
TF	Transription Factor
UTR	Untranslated Region
XAI	Explainable Artificial Intelligence

Chapter 0

Introduction

0.1 Scope of this thesis and brief statement of scientific advancements

In this thesis we focus on the development of High Performance Computing Machine Learning methods specifically devised for tackling problems in Precision Medicine. Data analysis and inference in this research field poses several unique challenges, as the volume of data is generally massive and the computational approaches for extracting biomedical knowledge from the data are extremely resource demanding. For this reason, new efficient ways of data processing must be conceived, and one of the most viable approach is represented by High Performance Computing techniques.

Our main contributions fit in this picture, as we propose three HPC - oriented approaches for open problems in Precision and Genomic Medicine: specifically, we propose a parallel and highly scalable Machine Learning - based solution for the prioritization of deleterious genomic variants; we also investigate how Deep Neural Networks can be used to assess the activity of the regulatory regions across different cell lines; finally, we propose a GPU accelerated method used for labeling nodes on a graph, a general technique used for modeling several problems in Precision Medicine, such as drug repurposing or automatic protein function assessment.

We hope that the High Performance Computing Machine Learning methods proposed in this thesis will represent valuable tools to support research in Precision Medicine.

0.2 Structure of the thesis

This work is structured as follows: in Chapter 1 we introduce Artificial Intelligence methods for Precision Medicine. We discuss the three main factors that in recent years contributed to its rise in popularity: Next-Genereation Sequencing techniques, Big Data and Machine Learning. As one of the main focus of this thesis is the last of the three, we briefly provide an insight of its specific application in Precision and Genomic Medicine, with attention to the past and current research in variant pathogenicity prioritization and regulatory region activity prediction. We also discuss some important critical issues and open challenges in Machine Learning for Precision Medicine.

In Chapter 2 we summarize the main reasons behind the need of High Performance Computing (HPC) and the main challenges that this paradigm tries to solve. We present the hardware and software technologies which are currently available. Hardware wise, we start the dissertation from multi- and many- core architectures, up to large supercomputers. Software wise, we illustrate the current frameworks that partially leverages the difficulties in programming these parallel architectures. Finally, we briefly discuss the history and state of the art of HPC applications specifically developed for use in Precision Medicine and Machine Learning.

In Chapter 3 we present *parSMURF*, a novel HPC Machine Learning method for the prediction of deleterious and pathogenic Single Nucleotide Variants in the non-coding area of the genome. This binary classifier has been specifically developed for dealing with highly skewed dataset, where one class is orders of magnitude larger than the other: this makes the classification task highly difficult and many traditional approaches fail. We provide a fast parallel and scalable implementation which improves the original algorithm in both execution speed and accuracy of the results. We also present *ParBigMen (parSMURF application to Big Mendelian data)* an extension to this work that was recently awarded by 50M computing core-hours by the Partnership for Advanced Computing in Europe consortium (PRACE). Through the use of the newly developed *parSMURF-NG* and by leveraging HPC facilities, we plan to evaluate new genome-wide Regulatory Mendelian Mutation (ReMM) scores for the prioritization of non-coding variation in the human genome, which are going to be released in the next version of *Genomizer*, providing a invaluable support for the entire scientific community working in this field.

In Chapter 4 we present a GPU accelerated Deep Neural Network model for the assessment of the regulatory region type and activity across different cell lines. Specifically, we developed a Feed-Forward Neural Network and a Convolutional Neural Network that leverage epigenomic and sequence data, respectively, for identifying the activity of regulatory regions in the non-

coding area of the genome. We show that our approach is able to detect enhancers and promoters that regulate specific cell lines and, more in general, is able to distinguish tissue-specific regulatory regions. We also show that different dataset rebalancing techniques, as recently used in literature, may influence the outcome of the approach. Finally, through the use of Bayesian Optimization and a parallelization scheme, we show how the execution time can be substantially reduced and, at the same time, the quality of prediction can be improved.

Finally, in Chapter 5 we present *ParCOSNet*, a multi-parametric Hopfield network -based GPU accelerated method for labeling partially labeled graphs. This is a general problem that frequently comes up in Precision Medicine, for example in pharmacogenomics, drug repurposing or the Automatic Function Prediction problem. Our solution uses a co-operative CPU-GPU scheme that substantially decrease the computation time, but keeps the Hopfield dynamics sequential by splitting the evaluation over several Maximal Independent Sets of the underlying graph. Also, thanks to its efficient memory allocation scheme, *ParCOSNet* is capable to process very large datasets, thus making this approach suitable for the Big Data era. We also present as extension of this work, a novel GPU accelerated model for solving the Maximal Independent Set problem over a graph, with a strong focus on balancing the class size. We provide a GPU implementation of the approach and results show that this new coloring strategy provides optimal class balancing with a computational time comparable to greedy strategies.

Chapter 1

Machine Learning for Precision Medicine

This chapter introduces Machine Learning for Precision Medicine, describing its definition, its goals, the current state of the art and the future objectives. The focus of this review is on Precision Medicine computational aspects, in particular on some relevant Precision Medicine problems discussed in this thesis. A strong focus is on deleterious and pathogenic variants prediction and the prediction of regulatory regions, both using Machine Learning methods.

1.1 Precision Medicine

Precision Medicine (PM) is a new paradigm of medicine which has recently become very popular. According to the 1999 foundational paper by Francis Collins "Medical and Societal Consequences of the Human Genome Project" [1], Precision Medicine will "completely transform therapeutic medicine".

However, it is very hard to provide a definition of Precision Medicine which encompasses its every aspect and all of its goals. According to the Precision Medicine Initiative [2], a consortium announced by former United States of America president Barack Obama in 2015 then renamed 'All of Us', Precision Medicine is defined as "an emerging approach for disease treatment and prevention that takes into account individual variability in genes, environment, and lifestyle for each person." This represents a major departure from the "one size fits all" practice in diagnosis and prevention of classical medicine, towards an approach that tries to find personalized treatment and prevention measures specifically tailored to group and subgroups of patients

on the basis of their genomic material, lifestyle, personal history and external environmental factors [3].

Also, another major difference from traditional evidence-based medicine is the importance of incorporating a vast and variegated amount of biomedical data - from bio-images to genetic material, but also physiological data acquired with wearable devices - as support in the prevention, diagnosis and treatment of illnesses: as a consequence, moving from the traditional analysis of observable signs and symptoms towards the efficient use of novel biomarkers, allows more precise diagnosis [4]. This is especially true, for example, for cancer diagnosis and treatment, as cancer is more increasingly seen under a molecular context rather than for its histology characteristics [5].

We can summarize these two pivotal concepts stating that Precision Medicine is a paradigm in medicine which aims towards the identification of two new taxonomies: the first one for better stratifying patients, finding sub-grups on the basis of common genetic traits, lifestyle, and environmental factors [4]; the second for better stratifying diseases: according to the National Research Council Committee in [6], the three fundamental goal of the definition of this new disease taxonomy are (by quoting):

- "Describe and define diseases based on their intrinsic biology in addition to traditional physical “signs and symptoms”"
- "Go beyond description and be directly linked to a deeper understanding of disease mechanisms, pathogenesis, and treatments"
- "Be highly dynamic, at least when used as a research tool, continuously incorporating newly emerging disease information"

The difficulty in accurately define Precision Medicine also comes from the fact that over the years it replaced the terms *personalized medicine*, *individualized medicine* or *stratified medicine* that still many authors use interchangeably as synonym [7]. Nowadays Precision Medicine superseded them: the argument here is that the term "personalized" could lead to the erroneous conclusion that Precision Medicine aims at creating treatments on individual level. However, since the dawn of modern clinical medicine, physicians treated each patient on an individual basis [8, 6]. The focus here is not on the individual patient, but in exploring how genetics, environment and lifestyle can be used to identify population subgroups which could be more susceptible to a particular disease, or could develop a particular prognosis or could react in a specific way to a specific drug, with the aim of targeting the most appropriate intervention strategy for the prevention, diagnosis and treatment of each diseases in each subgroup [9, 6].

Also, Precision Medicine differs from personalized medicine in the healthcare model defined by the two approaches: personalized medicine, being patient-centered, assumes a strong relevance in a patient preference, social context, lifestyle and knowledge, thus creating a strong bond between patients and physicians. Instead, we can consider Precision Medicine as more "data-driven", defining a model that goes beyond genomics and strongly relies on information, thus focusing in bonding patients, physicians, researchers and clinical laboratories, anywhere in the world [10, 11].

Finally, a common misconception comes from the fact that, despite a great deal of clinical information is extracted from genetic material, there is a tendency of confusing Precision Medicine with *genomic medicine*. However, Precision Medicine does not rely solely on genomics [12, 13]: besides the already cited lifestyle and environment factors, several biomedical data types heterogeneously contribute to the assessment of prevention, risk, diagnosis and treatment of illnesses. Examples of this biomedical data are not limited to bioimages (X-Rays, MRI, fMRI, CAT scans, etc), physiological data (for example coming from blood samples analysis), data collected by wearable devices (continuous monitoring of oxygen level in the blood, blood flow and pressure, etc), and so on. For this reason, it has been proposed that Precision Medicine should be considered as an umbrella term incorporating Predictive, Preventive, Pharmacotherapeutic (Pharmacogenomic and Pharmacogenetic) and Participatory Patient (Portable and Protective) components [14, 13].

In the following sections we will discuss the technology advancements that made Precision Medicine possible - with a special focus on Machine Learning - its current application in clinical research as well as future trends, and current challenges and criticism.

1.2 The technologies behind PM

There are three major driving forces behind the rise of importance of Precision Medicine:

(1) **The ability to quickly and cheaply generate a vast amount of biological and omics data.**

Since its commercial introduction in 2005, Next Generation Sequencing (NGS) revolutionized genomic sequencing, as it is a faster, more accurate and cheaper alternative than traditional Sanger sequencing method [15]. Nowadays, the cost for Whole Genome Sequencing is about 1000 dollars and the process takes less than an hour [16]. This ultimately led to massive collections of samples and whole population screening programs, like the One

Thousand Genomes project [17].

Besides the collection of DNA samples, technology advancements also allowed to substantially decrease the cost for collecting a wide variety of biological and molecular data commonly referred as *omics*: [18] names a few examples such as *transcriptomics* (the study of the expression level of each gene of an organism), *epigenomics* (the whole-genome study of epigenetic regulation), *proteomics* (the study of all the proteins in a cell or an organism), *metagenomics* (the comprehensive analysis of all the genetic material - DNA and RNA - of a patient sample comprising microbial, fungii, viruses and parasites) and many more.

Also, data generation and collection is not limited to molecular biology, but also biomedical data such as bioimaging [19], physiological data (ECG,...) and a whole other range of datasets related to manually curated evaluation by physician (Electronic Health Records). Lastly, we feel to include data collected from diverse and unexpected sources, as in the case of Social Media and mobile phones: an example comes from a 2013 study in epidemiology [20] in which queries to Google Search and Twitter related to disease symptoms have been used for detecting the spread of infective diseases [21, 22], or the recent work that, by tracing mobile, quantitatively estimated the main causes of COVID-19 spread across the population [23].

(2) **The ability to store, efficiently organize and distribute such a vast amount of data under the Big Data paradigm.**

Big Data is the term used for indicating the storage, process and distribution of data which is often beyond the capability of a single local computer due to constraints in memory size or processing power offered by a machine [21]. Datasets complying to these traits are informally associated with the "five Vs" definition:

- Variability (no inherent defined structure)
- Variety (includes heterogeneous data types)
- Velocity (requires high speed in both processing and transmission)
- Veracity (data may be noisy and uncertain)
- Volume (dataset extremely large in size for features, number of samples or both)

Big Data technology involves both hardware and software solution specifically designed to deal with such data. Hardware wise, data is stored in

server farms or data centers whose machines are qualified by huge memorization capabilities, a fast interconnection subsystem and high processing power. Software wise, data is managed (organized and made ready-available for access and redistribution) by solutions specifically made for Big Data, such as Apache Hadoop [24].

Datasets are organized and made available for researchers and physician: several free-to-use omics database were created in the last years. A few examples are the ENCODE project [25] which identified promoters and enhancers in 147 cell types using a wide range of high-throughput technologies; the FANTOM project, which employed CAGE (Cap Analysis of Gene Expression) technologies to broaden the spectrum of considered samples, including 1,816 human and 1,016 mouse samples [26]; the International Human Epigenomic Consortium [27], which aggregates several databases from seven consortia, making available regulomic, methylomic, and transcriptomic data from more than 600 cell lines.

(3) **The ability to process, extract the relevant information and create inferences on the basis of these data using modern computerized approaches.**

As the quantity of collected and ready-available data increased at an unprecedented rate in the last years, researchers and physicians felt the urge to find efficient methods for data processing. Aside from the volume of data, another issue that arose as a consequence of data-drive research is the ability to integrate several different data types (i.e. bioimages with genomic data) for increasing the level of detail that can be extracted by the conjunction of those data. The challenge here is to efficiently extract useful information from these multidimensional and articulate datasets [18].

Traditionally, ad-hoc mathematical and statistical models have been used: for instance, [28] describes a mathematical model for deciphering the signatures of mutational processes in cancer genomes. However, in recent years methods based on Artificial Intelligence (AI), particularly Machine Learning (ML), have been shown to perfectly fit in the data-driven Big Data approach as they solve both the problem of high-volume data processing and heterogeneous data integration.

Machine Learning is a popular sub-field of Artificial Intelligence [29] used for automatically detect patterns in data which often cannot be identified by hand. It leverages the use of very large dataset for training a learning algorithm capable of discovering previously unknown associations and interaction patterns among variables [30, 31]. Training of the learning algorithm follows an adaptive approach, but without explicitly programming sets of rules that may guide the learning process - as happens, for example, in expert sys-

tems [32]. The adaptive learning approach works iteratively by providing data to the algorithm which automatically adjusts its internal parameters (e.g. weights) by trial and error [21]. Once trained, ML models can be provided with new unseen data to make decisions such classification, inference or discrimination.

Data provided during the training phase can or cannot be associated with a label; this ultimately identify two macro-classes of learning machines: supervised and unsupervised learning algorithm. Supervised algorithm are mainly used for classification and regression, while unsupervised are used for clustering, data exploration and generation of new hypothesis [32, 33]. Examples of supervised learning algorithms are linear and logistic regressors, Generalized Addictive Model, Tree-based model such as Decision Trees, Artificial Neural Networks (shallow or deep), and Support Vector Machines; instead, k means clustering and Principal Component Analysis are examples of unsupervised models. We refer to [32] and [34] for an accurate dissertation of each of these learning methods.

Another classification of Machine Learning methods is based on whether features are manually or automatically selected and extracted from the dataset. Several ML algorithms such as Decision Trees and Random Forest rely on the availability and quality of hand-curated / selected features that characterize each sample of the dataset [29]. Features can be domain-specific (i.e. targeted to the problem to be solved) or domain-agnostic [21]; regardless of the type of features, its engineering is a delicate process that can affect the outcome of the learner [35]. On the opposite, Deep Learning [36] leverages the process of feature engineering by automatically learning the data representation from the data itself. However, although easier to use and more practical than hand-selected feature approaches, Deep Learning methods require a high volume of data for this process to be effective, hence it may be unsuitable for small sized datasets [37].

In recent years, fueled by the rise of Big Data and the availability of large dataset, ML has enjoyed a relevant success in various fields such as financial, social, insurance, security, home automation; it found its way in every level and scale of business and society, from big companies like Google and Facebook, to small businesses. Application types range from highly sophisticated such as self-driving cars or face recognition, to more mundane like photo image altering. Lastly, thanks to the diffusion of easy to use ML and DL frameworks and hardware [38, 39], even small companies could afford and implement a ML-driven solution for their business, as in this case where a Japanese farmer uses Deep Learning for sorting cucumber depending on their visual quality [40].

Machine Learning techniques have been been extensively used in medicine.

To cite a few examples, successfully applications have been created in the field of bio-imaging for the detection of abnormalities by analyzing X-Rays images [41] or assessing the risk of aneurysm ruptures using CT scans [42]; on clinical monitoring, [43] uses vital signs to forecast cardiovascular instability; in [44] a Deep Neural Network trained on Electrocardiograms (ECG) is able to predict better than trained cardiologists irregularities such as arrhythmia; in [45] ML is used to model the heterogeneity of CD4+ T-cells.

For more specific dissertation on Machine Learning techniques used in genomics, particularly for investigating its non-coding regions, we refer to section 1.4.

1.3 How Precision Medicine is reshaping clinical practice

One of the most relevant way of performing patient stratification is through diagnostic and prognostic biomarker discovery. It is a well known fact that the common presence or absence of a biomarker in a sub-group could be the indicator of the potential risk of developing a disease in the future, or of the presence of an on-going pathology. Specifically, according to [13], biomarkers can be divided in four groups:

- Disease prevention biomarkers, used to identify individuals and groups of individuals at risk of developing a condition
- Diagnostic biomarkers, used to distinguish between conditions
- Prognostic biomarkers, for assessing the prognosis of a disease (can be used during diagnosis and prognosis for deciding the intervention steps)
- Disease progression biomarkers, that delineate the underlying mechanism of disease progression

The presence of a genetic mutation, the detection of abnormal levels of a specific protein, as well as cytological and histological characteristics are well know biomarkers, but also gene expression or mRNA alteration have also surfaced as valid alternatives. Also, Next-Generation Sequencing allowed the discovery of new biomarkers related to cancer such as mutation signatures or tumor mutational burden [46].

While there are several excellent reviews profiling the current status of biomarker discovery, a deep report of the current state-of-the-art of this field is out of the scope of this dissertation. As a brief reference, [47] shows several examples of Big Data-driven precision medicine approaches that helped

finding biomarkers in motion system, circulatory system, respiratory system, urinary system, endocrine system, digestive system, nervous system, reproductive system and cancer; [10] reports similar findings applied to Diabetes, particularly in patients having rare genomic variants; [13] in Multiple Sclerosis; [48] in pediatric oncology.

Traditionally, biomarkers discovery has been mechanism-driven, i.e. studying and understanding the molecular mechanisms underlying biological processes, which has the advantage of giving an actual insight on the causes and effects of the biomarker itself. However, identifying biomarkers is challenging: the volume of whole-genome and whole-exome data is growing very fast and, although this provides an ever growing source of information of molecular activities, computational methods are nowadays required for process this amount of data [49]. Aside the traditional way that relies on hand-crafted mathematical and statistical model, nowadays the preferred way of computationally discover biomarkers is through the usage of Machine Learning methods [50, 51, 52, 53, 54].

Biomarkers identification is only one side of Precision Medicine, as drug development and repurposing play an equal important role in reshaping clinical practice. In the context of Precision Medicine, pharmacotherapy involves the use of predictive methods to precisely select treatments and drugs dosage. In this context, pharmacogenetics is the field of research that aim to identify patients subgroups that are more likely to respond to a specific treatment or experience side effects, based on their genomic profile and presence or absence of relevant biomarkers. Also, pharmacogenetics aims at identifying novel drugs and providing insights about the functional characteristics of the expression of a gene inside a specific cell type or organism [10, 5].

The landmark study in this field is the one that ultimately led to the discovery of the imatinib drug for the treatment of chronic myelogenous leukemia (CML) [55, 5, 3]. The observation in the late 1960' that the 95% of patients with CML has what has been called the *Philadelphia chromosome* led to the discovery of the BCR-ABL biomarker. This ultimately led in the late 1990' to the development of a drug which directly blocks the activity of the BCR-ABL fusion protein. Its relevance is remarkable because it was the first drug that was engineered for targeting a specific protein. Amusingly, this discovery was worth the Time magazine front cover, labeled as "bullet" against cancer [56].

Again, a comprehensive review of the state-of-the-art of pharmacogenomics is not in the scope of this work, and we refer to [57, 58, 59, 3, 60].

Briefly it is worth citing that another aspect of pharmacotherapy in Precision Medicine is related to drug repositioning, i.e. finding possible new

therapeutic purpose for already existing and available drugs. This is dictated by the fact that the development of a new drug is usually a very long process (requiring also several trial stages before clinical adoption) and requires huge monetary investments by pharmaceutical companies. By leveraging omics data and Machine Learning techniques, it is possible to discover new applications for drugs, as reviewed in [61].

Lastly, a few other new applications that are not omics-related but falls under the Precision Medicine umbrella (as involving intensive data collecting and processing) are related to health monitoring: for example [62] shows how Chronic Obstructive Pulmonary Disease can be early detected by using a respiratory sensor, telemonitoring and a ML algorithm, while [63] shows how seizures can be detected using a wrist accelerometer, telemonitoring and a DL approach; health related Apps [64, 65]. Finally, it is worth noting the application of Natural Language Processing for summarizing and extracting information from Electronic Health Records or from the huge amount of scientific literature available online (literature mining) [46, 60].

1.4 Machine Learning in Precision Medicine

As briefly illustrated in section 1.2, Machine Learning has been successfully applied in several studies related to Precision Medicine. In particular, the field of genomic medicine particularly benefits of this approach: as an example, in the context of genomic analysis, Machine Learning methods have been developed for recognizing the transcription start site in a genomic sequence [66], as well as specific functional motifs such as promoters and enhancers [67]. In this section we want to focus our attention on two promising application of Machine Learning methods to Precision Medicine and how Machine Learning could be leveraged as a powerful tool for the analysis of omics data.

1.4.1 Prediction of Deleterious Variants

One of the most important goal of research in human genetics is the identification of DNA variants that impact biomedical traits, with a focus on those that are causative of deleterious conditions or pathologies, as these can deeply explain the biological mechanism underlying many common and rare diseases. In the last 25 years, after the conclusion of the Human Genome Project [68] and thanks to all the subsequent technology improvement - especially those related to genome sequencing, comprising whole exome and whole genome sequencing - researcher were able to identify many genes and

variants that are causal of diseases, from Single Nucleotide Polymorphism to complex structural rearrangements of genetic material [69].

Initially, researchers were focused on the identification of causative variants occurring in the coding area of the genome: this, for example led to the discovery of the genetic cause of Huntington's disease [70, 71]. This was justified by the fact that until the late 2000's all the non-coding regions of the DNA (around the 98% of all the human genetic material) was assumed not to have any biological function, therefore being classified as 'junk DNA', hence variants and mutation occurring in these regions were thought to be irrelevant and not to be causative of any medical condition [72]. However, the ENCODE project and subsequent studies unraveled that the non-coding regions of the DNA also have biological functions: it was shown that, for example, they regulate the transcription of genes, as many of these areas are accessible by transcription factors [25]. This evidence of functionality changed the researchers' point of view, because now it is well accepted that variants and mutations occurring in non-coding regions could be causative of pathologies, since they may interfere with the transcription and gene expression mechanisms [73, 74, 25].

In this context, one of the most used experimental design for the assessment of the correlation between genomic variants - in particular, Single Nucleotide Polymorphism (SNP) - and biological traits in samples from a population, are the Genome Wide Association Studies (GWAS) [75]. In this protocol, the DNA of subjects within a population having a common phenotype is compared; through the use of SNP arrays, millions of common SNP are read; finally, SNP are ranked by frequency, so that the most frequent variant is then said to be associated with the phenotype. SNP are also ranked taking into account the SNP coming from a control group that does not show the observed phenotype [76]. The underlying computational model of GWAS studies is mainly statistical, and calculation is usually performed through specialized tools such as *PLINK* [77], *ProbABEL* [78] or *SNPTEST*. A comparison of several statistical methods for imputation-based association methods is available in [79].

The first GWAS study was published in 2002, showing that SNPs in the lymphotoxin-alpha gene are associated with susceptibility to myocardial infarction [80]. Nowadays, successful GWAS studies are in the thousands: a few more recent examples are [81, 82] for cancer or [83] for Alzheimer, while countless excellent reviews have been published in recent years [84, 85]. Despite all these studies, GWAS shows several limitations, the most important being that the identification of a locus does not automatically imply a causal correlation with a phenotype and additional steps are required to identify the exact causal variant and, if the locus falls in a non-coding

region, the corresponding target gene(s) [85].

In recent years, however, a more data-centric approach for the imputation and prioritization of genetic variants has surfaced, and at the heart of this approach are Machine Learning methods. Thanks to data integration and powerful learning strategies, these methods are able to prioritize variants and also predict the impact of SNPs in terms of deleteriousness and pathogenicity across all the genome.

From the late 2000's, several methods were developed specifically for this aim but, as assumption that non-coding regions of the DNA have a biochemical impact came only in the early 2010's, the first methods were essentially focused on evaluating the impact of SNPs occurring in the coding regions. One of the first approach was the *SIFT* (Sorts Intolerant From Tolerant substitution) algorithm [86] that exploits sequence homology and an ad-hoc mathematical model for evaluating if an aminoacid substitution affects protein function. It was followed by *PANTHER* [87] that uses an Hidden Markov Model to rank SNPs missense according to the likelihood of impacting protein function. *PhD-SNP* [88] leverages Support Vector Machines trained with protein sequence information for predicting the pathogenicity of non-synonymous coding SNPs. Similarly, *nsSNPanalyzer* [89], but using a Random Forest classifier. Also *SNAP* [90] was designed to evaluate the deleteriousness of non-synonymous SNPs, but its relevance is in the fact that was one of the first methods exploiting feed-forward neural networks. *MutPred* [91] also uses a SVM classifier, but in tandem with a Random Forest for evaluating the probability of gain / loss in protein function between non-mutated and mutated protein sequences. *PolyPhen-2* is one of the earliest most important algorithm as it uses a different set of heterogeneous features for performing the classification task, as both sequence-based and structure-base features are used [92], while the classifier is Naive Bayes. Finally, *SNPandGO* [93] is an interesting SVM-based method trained with features from sequence and Gene Ontology.

More modern approaches, but still tied to the coding regions of the genome are *REVEL* [94] which basically is a Random Forest classifier trained with 18 pathogenicity predictor from 13 different tools; *MISTIC* [95] is another ensemble learner based on Random Forest and Logistic Regression for the evaluation of the impact of missense variants. A similar approach has been developed in *ClinPred* [96], while *PrimateAI* [97] uses Convolutional Neural Networks trained with sequence data for the prediction of the impact of missense variations.

After the realization that non-coding regions of the DNA carry biological functions, several Machine Learning methods were developed to overcome the limitations of all the tools available at the time, as they were designed to deal

only with variants occurring in the coding regions. However, this task proved to be very challenging for several reasons including the fact that the number of neutral variants is far larger than the pathogenic and deleterious ones; also, the at-that-time understanding of biochemical mechanism and splicing were a relevant support for the identification of candidates, but non-coding regions were - and still are - relatively new, hence many of the underlying mechanisms are still to be unraveled.

The first and even today most important tool specifically design for this aim was *CADD* [98]: its strength lies in the use of a single annotation comprising several features - the most relevant being evolutionary conservation scores - for training a Support Vector Machine. In the latest version, the classifier was changed in favor of logistic regression [99]. CADD represented a breakthrough in the assessment of the impact of variant, and was followed by several other works. *DANN* [100] exploits CADD set of features for training a Deep Neural Network. *GWAVA* [101] uses a modified Random Forest classifier trained with a set of features comprising conservation scores and biological features. A similar approach was used in *FATHMM-MKL* [102], using a classifier based on multiple kernel learning, and in *LINSIGHT* [103] using a Generalized Linear Model. *Eigen* [104] proposed a different approach, as it used unsupervised learning - i.e. training a classifier using an unlabeled dataset.

A radically different approach is implemented in *DeepSEA* [105], *DeepBind* [106] and *gkm-SVM* [107], as they do not rely on a set of biological or evolutionary features, but they train their classifier (Deep Neural Networks for DeepSEA and DeepBind, and a gapped k-mer Support Vector Machine for gkm-SVM) using raw DNA sequences.

Finally, the most recent approaches are *ReMM* [108], *hyperSMURF* [109] and *NC-BOOST* [110] that leverage Random Forests (ReMM and HyperSMURF) and gradient tree boosting (NC-BOOST) for scoring pathogenic non-coding SNP variants in the context of Mendelian diseases. They all rely on a set of features similar to CADD, but all the classifiers were trained with a set of manually curated known pathogenic or deleterious variants as positive class. hyperSMURF also addresses the problem of class unbalancing, as known annotated Mendelian SNVs in the non-coding regions are vastly outnumbered by neutral variants and, at present times, it is still the state-of-the-art for the prediction of Mendelian SNVs in highly imbalanced datasets.

In this thesis we address the computational problems of hyperSMURF, presenting an highly parallel approach to overcome the computational complexity of the algorithm.

1.4.2 Regulatory region prediction

Another important field of reasearch in precision medicine that greatly benefits from Machine Learning, is the identification of regulatory sequences across the human genome. These are region of the non-coding DNA that regulates gene expression: in particular, Cis-Regulatory Elements (CREs) are regulatory regions that control the expression of neighboring genes [111]. These elements play different roles in gene expression and can be grouped depending on their function; the most well studied and characterized are Enhancers and Promoters: the latter are regions very close to the gene to be transcripted and include the region where transcription is initiated [112], while the former are regions which can be either close or relatively far from the transcription site that influence (enhance) the expression level [113].

It is now clear that these regions play a crucial role in gene expression. For this reason, it is particularly important to identify them and correctly understand their core roles in gene expression. This is particularly relevant when dealing with diseases: several studies showed that variants and mutations in these regions are associated with both rare and common diseases [114, 115, 116].

One of the key characteristics of promoter and enhancer is that they possess different *activation statuses* depending on the cell types [117, 118]: on the other way around, we can say that it has been shown that one of the key mechanisms for cell type differentiation is the activation status of promoters and enhancers [119]. For this reason, understanding their activity in specific tissues provide a great insight on the influence of, for example, SNPs CREs, as it is highly probable that variants occurring in active enhancers or promoters are more likely to be associated with pathogenicity of a condition than those occurring in their non active counterparts.

The identification of putative cis-regulatory elements has been largely accomplished in several experimental ways, being classical genetics or, more recently, through the use of biochemical methods [120, 121]. Such methods involve the detection of chromatine features as well as histone modification which are associated with functional non-coding regions for inferring candidate cis-regulatory elements. Thanks to a fast and consistent decrease of the cost of these in-vitro analysis methods, several consortia, including ENCODE [25], collected and aggregated the results of these biochemical assays into large publicly available databases which now contain over one million putative enhancers over several cell lines. These resources allowed researchers to develop new in-silico methods for the identification of CREs, as well as their tissue specific activity level and the possible impact of variants occurring in them. A list of currently available databases is available in [120],

Table 1.

Among the first proposed work in this field, *ChromHMM* [122] is a seminal computational method for the prediction of chromatine state. For this task, it leverages a multivariate Hidden Markov Model trained with several chromatine marks; in 2012, it was one of the first methods that characterized the biological function of cromatine states, correlating the results with large-scale functional datasets. Following ChromHMM, *Segway* [123] is another unsupervised method employing Dynamical Bayesian Networks trained with epigenetic features such as the positions of histone modifications, transcription-factor binding and open chromatin for identifying transcription start sites, gene ends, transcriptional regulator CTCF-binding regions repressed regions and, most important, enhancers. Also in 2012, [124] proposed a Random Forest -based method trained with so called Transcription-Related Factors from the ENCODE database for the identification of regions with active or inactive binding, regions with extremely high or low degrees of co-binding and finally regulatory modules proximal or distal to genes; this in turn is used to infer putative enhancers which have been also experimentally validated.

Starting from this work, over the years several ML-based methods have been proposed for the identification of enhancers: the following are presented chronologically and represent only the most relevant methods among the ones that have been published. In 2013, *RFECS* [125] employed a classifier based on Random Forest trained on genome-wide profiles of 24 histone modifications in two distinct human cell types, embryonic stem cells and lung fibroblasts for the prediction of enhancers in different cell lines. *DEEP* [126] (2014) uses an ensemble of Support Vector Machines trained with features derived from histone modification marks or attributes coming from sequence characteristics; it is able to correctly identify enhancers across different cell lines. *DELTA* (2015) [127] uses an AdaBoost classifier trained with histone modification features. Also in 2015, *dREG* [128] uses a support vector regression model trained with features from global run-on and sequencing (GRO-seq) for identifying enhancers, promoters and insulators. Few more methods proposed in 2016 are *eRFSVM* [129] which uses a hybrid SVM and RF classifier trained with features from ChIP-Seq datasets and labels from the FANTOM5 database [26]; *EMERGE* [130] is an integrative framework that leverages a classifier based on logistic regression for the identification of enhancers, and the novelty lies in the fact that features are heterogeneous in nature, being extracted from different data types such as ATAC-seq, ChIP-seq and conservation scores. *REPTILE* [131] (2017) is another method based on Random Forest trained with features based on tissue-specific local epigenetic marks such as DNA methylation and histone modification profiling. A similar approach has been used in *EAGLE* [118] employing AdaBoost as base learner;

the focus of this work, however, lies in the fact that the classifier is trained with a very small number of features. The idea behind this choice is that, being the roles of enhancers and promoters tissue-specific, it may be possible that not all the features are available on a wide range of tissues, hence providing a small number of very descriptive features may be sufficent for correctly discriminate among a wide number of different cell types. Finally, *PREPRINT* [132] (2020) uses a probabilistic Bayesian approach.

As previously stated, Deep Neural Networks were a major game changer in Machine Learning as they introduced notable breakthroughs in data processing, data analysis and statistical inference. As a consequence, several DNN-based methods were specifically developed for the identification of CREs. One of the key advantage of DNN methods compared to other Machine Learing techniques, is that there is no need to manually curate the sets of features, as data representation is automatically learned from the data. In this context, it means that, for the first time, CREs could be identified by the raw DNA or RNA sequence itself, without relying on other biochemical assays. The following is an list of the most relevant methods, presented in - almost - chronological order.

DeepBind [106] (2015) was one of the first DNN models to predict binding sites from sequence specificities of DNA- and RNA-binding proteins. According to the authors, its Convolutional Neural Network (CNN) model was trained with 12 TeraBytes of sequence data, and outperformed any other at-the-time available method for the prediction of binding sites. Its novelty paved the way for a deeper exploration of the subject: to this end, *PEDLA* [133] (2016) provided a more specific approach to enhancer identification. Its model was trained with a wide range of heterogeneous features derived from ChIP-Seq data included 27 histone modification marks, 15 TFs and co-factors, but also chromatin accessibility and transcription obtained from DNase-Seq and mRNA-Seq, and finally features comprising DNA methylation, CpG islands and evolutionary conservation. The underlying model is a 1 or 2 hidden layer Feed-Forward Neural Network (FFNN). Also in 2016, a similar approach was used in *EP-DNN* [134], as a deeper 5-level FFNN was trained on epigenomic data based on four histone modification dataset. More similarly to DeepBind, *BiRen* [135] (2017) attempts to predict enhancers solely by using raw sequence data. Its model is based on a hybrid approach, as CNN and gated recurrent unit (GRU)-based bidirectional recurrent neural network (BRNN) modalities are used for this task. Also *DeepEnhancer* [136] (2017) has been similarly proposed, but only employing CNNs. One of the latest work which proposes a model for CREs identification is *Aikyatan* [137] (2019) which uses a conjunction of SVM, RF and CNN models trained with epigenomic features for the identification of Distal Regulatory Elements.

All the methods that have been presented so far have as a common trait the fact that they are specifically designed to identify whether a non-coding region is a CRE. However, as previously stated, CRE are characterized also by tissue-specific activation status which is particularly relevant when evaluating, for instance, the impact of a SNP in a CRE specific to a cell type. For this reason, in the latest years, the focus of the research community shifted towards the characterization of the regulatory region activity level across cell lines, and recently several new methods have been proposed. *Basset* [138] (2016) is the first step towards this goal, as its CNN network trained with DNase-seq features is able to correctly predict the accessibility of DNA sequences (chromatine state) across a wide range of cell tipe. More recently (2018), Basset was improved and re-released as *Basenji* [139] and the main improvement lies in its learning model, as the CNN now accepts raw sequence data and is able to correctly predict promoters and distal regulatory elements. *DECRES* [140] (2018) was recently proposed and used a FFNN model with three hidden layer and trained on epigenomic features for identifying promoter and enhancers, together with their activation states as either active or inactive in a specific tissue. Two of the latest works in this field (2019) are *ChromDragoNN* [141] and *MPRA-DragoNN* [142]: the former is a method for predicting genome-wide chromatine accessibility based on Basset, but its strength lies in its multi-modal model, as both expression and sequence data are used to provide a better outcome, since both data types provide different information that are used efficiently by the Residual Neural Network learner; the latter, is able to predict and interpret CREs activity from massively parallel reporter assays (MPRA) data. Also *iEnhancer-ECNN* [143] (2019) is an ensemble of CNN for identifying enhancer together with their activity starting from raw sequence data.

Finally, during the last year, new integrated frameworks for leveraging many cumbersome tasks were released. These frameworks are modular by design and comprise different interacting parts for data downloading from publicly available catalogs, data pre-processing and merging, and also feature several learning modules ranging from classical machine learning (SVM, RM, and so on) to DNN. These frameworks are able to handle the CRE identification and activity prediction tasks. Two of the proposed frameworks are *Janggu* [144] and *Selene* [145].

In Chapter 4 we propose a Deep Neural Network model for the prediction of the regulatory region activity across different cell lines. Our research is aimed to expand the work of [140] and we developed a model based on a FFNN trained with epigenomic data and a CNN trained on raw sequence data.

1.5 Critical issues and challenges

1.5.1 Critical issues in Precision Medicine

Despite the astounding technical advancements that in the last 20 years turned Precision Medicine from a bright promise into an actual tool for clinical research, there are still many open issues and challenges that need to be addressed before Precision Medicine could be mass introduced and accepted into clinical practice.

Many authors cited data privacy as one of the main concern of the consequence of mass data collecting: this is an issue that transcends genomic and precision medicine, as the issue of compulsive data collection is proper of the Big Data paradigm. Privacy, ownership and control of genetic information, and consumer use (and even abuse) are still partially regulated or completely unregulated. Several authors urge the need of debating and regulating genetic privacy, as each individual DNA is unique and proper to that subject: DNA cannot be changed as if it were a password, a phone number or an address. Also regulation of this instance means that consumer should be properly informed how the collected genetic material could be used. Possible unauthorized use of this piece of information can influence, for example, health insurance fees or could lead to discrimination in job selection processes [21, 146, 147]. This matter is further complicated by the fact that different countries generally have different approaches and regulations to data privacy.

Another interesting issue involving the ethics of Precision Medicine is related to the possible source of discrimination in the process of patient stratification. The observation comes from the fact that increasing stratification increases the chances of having subgroups with reduced sample size, hence only certain subgroups will be considered for clinical trials. This is particularly true in subgroups characterized by rare biomarkers where uncertainty of treatment effect is particularly high [148, 149].

Several authors compare the effectiveness of Precision Medicine to more standard approaches, questioning if the effort of investing time, energy and money in the development of Precision Medicine -based treatments is worth the investment. In [21], the author noted that changes in environmental factors and change in lifestyle related to diet and exercise may be more effective and more relevant than genetic factors in several diseases [150] - also far away more economical than developing a new drug that only a small minority of very wealthy people would benefit. Also, Since the National Center for Human Genome Research was founded in 1989, life expectancy in the U.S.A. increased by four years, and none of the reason was tied to genomic advancements: this increment is more justified by, for example,

development of more effective vaccines, increased tobacco restrictions and control, or improved screening programs for common cancers. Also, these interventions were targeted to a major segment of population, often at a very low cost for public and private health companies [151]. As a side effect, this is very indicative of how a proper cost-benefit evaluation should be included not only in Precision Medicine, but in every field of research.

Strictly related to the economic factors and ethics, Precision Medicine could exacerbate the disparity between patients, as genomic treatments are still some of the most expensive on the market [10]. Moreover, as consequence of patient stratification, [3] states that patients that benefit from precision oncology are those carrying specific sets of variants, and usually those are in a very small subset of population [152]. Regarding the economic interest of pharmaceutical companies and the consequent disparity between patient, [48] reports that in pediatric precision oncology there is a general lack of interest in developing drugs that target genes mutated in common pediatric cancers, and the reason - quite sadly, however understandable - is that there is not an acceptable profit for developing such drugs, and the priority goes towards more common type of cancers such as lung, breast os colorectal.

Other minor issues and critical aspects are related to the trust of physician and patients towards PM methods [22, 46] that ultimately contributes to hinder the regulation and widespread inclusion in clinical practice, the liability of physician when adopting AI-driven PM methods [153] and reimbursement of PM practices by national and private health agencies [22, 154].

However, the glass is half full, as several of these issues and criticism can be addressed by discussion between physicians, researchers and lawmakers, and by introducing adequate regulations so to insure the staples of equity and equality proper to correct ethics.

1.5.2 Critical issues in Artificial Intelligence and Machine Learning

Several challenges and open issues are present in the context of Artificial Intelligence and Machine Learning, too: most of them are not exclusive to Precision Medicine and can be extended to other settings. However, data analysis in Precision Medicine poses several challenges, as biological systems are very complex and current models fails to correctly explain their complete dynamics: until a few years ago, it was thought that a few gene variants could explain the risk of complex and common diseases. However, several GWAS studies showed that diabetes, hypertension or obesity - to name a few - are connected to hundreds and sometimes thousands of gene variants

that alone (or even as a whole) fail to explain the actual risks and frequency of these diseases [155].

Another issue that is frequently associated with AI and ML is the lack of model explainability and results interpretability as they are often perceived as 'black boxes' - i.e. it is uncertain how a trained system produces a prediction - hence, it is very difficult to extrapolate which features are the 'most important', i.e. which feature(s) ultimately drive the outcome [154, 156, 157]. This may not be a problem in some context - for example Optical Character Recognition - but in clinical research often the 'why' is as important as the outcome itself. Also, there is a high chance that a lack of interpretability in this instance could lead to erroneous predictions due to artifacts or confounding effect which are not the actual cause of an observed behavior [158, 159]. For this reason, when identifying relevant biomarkers for a certain disease it is mandatory to filter items that show a strong correlation factor but lack causative effects. Unfortunately, this process cannot be automated, as in this field, knowledge is driven by experts that manually annotate data and evaluate results, thus strongly limiting scalability, as bound to human processing [46].

Quality of data is another issue: there is a common perception that quality of data greatly varies between the available sources [157, 154, 22] and this ultimately complicates data integration between sources and between different data types, especially in the context of multi-omics analysis [156] where better standards for data generation and reporting are required [160, 161].

In the Big Data age, data is compulsively and constantly collected; for this reason, data quantity is also perceived as an issue for AI and ML, mainly for two reasons. First, sometimes datasets are incomplete due to inconsistency or inaccuracy in data collection; however, missing data is not random data and its inference must be precisely evaluated. Several attempts have been proposed for the imputation of missing data such as [162] or [163] however this issue is still open as wrongly imputing missing samples could create biases in the dataset and lead to wrong or misleading predictions. Second, ML algorithms require a big amount of labeled data during the training process and often they perform poorly if not enough data are available. This is the case, for example, in datasets for extremely rare diseases were little information is available, or when discriminating between classes showing an abnormal disparity in size, for example when classifying a deleterious known genetic variants among the sheer amount of benign variants in the non-coding portion of the genome. The lack of labeled data which, again, must me manually curated, limits the scalability of the ML approch [157]. The drawback of dealing with high unbalanced datasets is the inherent bias towards one class, hence it is highly probable the AI algorithm during training only learns sam-

ples belonging to the most represented class.

Another major issue is the lack of validation and standardization in the application of AI and ML methods [46, 157]. This may affect the evaluation of methods and its adoption in clinical research. A classic example is the multiple ways for the evaluation of the performance of a learning algorithm, as there are several indexes for assessing how good a ML strategy could be. AUROC is a highly adopted index, but fails to measure a classifier performance if the dataset is highly unbalanced towards one class, thus reporting only the AUROC index could be highly misleading [156]. In this case, the evaluation should be performed using indexes that take into account class disparities, such the AUPRC index, as highly more informative than AUROC [164].

Finally, computational costs are also a big issue. In particular, the rise of the importance of Deep Learning models in the last ten years is also due to the widespread adoption of hardware accelerators such as Graphic Processing Unit (GPU) that allowed the training of fairly complex network without relying on server farms or other expensive specialized hardware. However, often the complexity of the model or the size of the dataset pose a strict limit on the size of the problem that can be addressed with AI methods.

As for the PM challenges, many of the proposed issues can be solved in the near future by selecting appropriate guidelines (data quality, validation and standardization). [154], for example, proposes the creation of accurate reference datasets by experts in a certain application field, enforced by FDA regulation for the evaluation of learning methods. As for the Interpretability of the model, several attempts have been made to "crack the black box", but they were case-specific and not very well generalizable. A more viable approach is to invest time and scientific effort into the Explainable Artificial Intelligence (XAI) paradigm which aims at the development of methods that, for example, ranks the dataset features to the predictor outcomes [165]. More technical problems are also being faced: in this work we propose three different accelerated ML strategies for overcoming the limitations imposed by the complexity of the underlying model. Also, in chapter 3 we propose a state-of-the-art model for dealing with highly imbalanced datasets.

Chapter 2

High Performance Computing and its application to genomics

In this chapter we summarize the current trends in High Performance Computing, focusing on accelerated and parallel algorithm for Personalized Medicine, Bioinformatics and Machine Learning.

2.1 The rationale behind High Performance Computing

As stated in chapter 1, one of the staple of Big Data paradigm is the extreme high volume of data and the consequent exceptional computational power required for its storage and processing. Informally, the term Big Data is used when a problem is represented by datasets that are unmanageable by standard computational means, hence requiring special technology to store and retrieve information from these, or extract some inference by applying statistical and numerical methods on these. As Big Data rose in popularity, concurrently the scientific community (but also the industry) felt the need to rely on tools able to cope with the massive computational power needed by the Big Data paradigm.

High Performance Computing (HPC) is an umbrella term that encompasses several aspects of this problem, as it both identifies:

- the set of hardware and software tools capable of reaching extremely high computing performance, generally by aggregating a large number of computing devices that individually or cooperatively work for the solution of a problem, hence including and superimposing several aspects of *parallel computing* [166]

- the ability itself to process information at a very high speed and volume [167]

Big Data is not the only reason for the existence of HPC, as the development of special computing techniques was traditionally driven by the need of solving problems that could not run on a single computational unit, for time or memory constraints (or both). For example, historically, parallel computing techniques have always been used for increasing a problem size - for example using larger dataset that cannot be stored in the memory of a single machine - or decreasing the computational time required for reaching a valid solution - for example, when the complexity of a problem is so high that cannot be tackled in a useful time span by a single processing unit.

In recent years, several technologies were developed to satisfy the ever increasing demand of computational power: nowadays, the most mature and used are multicore CPUs, hardware accelerators (co-processors) and massively parallel computing. In the following section we provide a brief illustration of each technology, as in this work they are all used - sometimes in conjunction - to tackle different problems. We point out that we cover the hardware and software solutions for data processing, leaving out all the techniques for data storage and retrieve. Also, special frameworks like Apache Spark [168] and Apache Hadoop [169] are just briefly discussed here, as they are designed to hide the parallelization layer through an abstraction process, focusing more on the usability of the framework itself, rather than the efficiency of execution.

2.2 Current trends in high performance computing

2.2.1 Multicore CPU and emerging architechtures

It is a very well known fact that, since the mid-2000, Moore's law [170] failed to hold up. Stated for the first time in 1965, it predicted that circuital complexity of integrate circuit - measured as number of transistors - would double every two years. Thanks to larger die size and increase in the miniaturization, in 1975 the prediction was true [171] and circuital complexity was also associated with computing power in terms of FLOPS of microprocessors and clock speed (measured in Hertz). This trend, however, held for about thirty more years, before microprocessors manufacturers impacted two physical barriers: limit to miniaturization and limit of power dissipation [172]. These barriers effectively stopped the Moore's law trend.

Around 2000, Intel partially circumvented this limitation by introducing CPUs featuring more than one processing core. Under this architecture, two or more threads / processes (depending on the number of available cores) can be executed simultaneously. This shift in CPU architecture forced a major change in software design, as to effectively exploit the full computational power of multi-core CPUs, programmers should embrace the parallel programming paradigm in the form of multi-thread software design. To this end, in recent years, several fairly high-level libraries were proposed to leverage a programmer the hurdle with low level thread functions: among them, Open Multi-Processing (OpenMP, in short) [173] is a C API comprising compiler functionalities and directives for almost automatically distributing a program workload over the available cores of a CPU.

A more low level approach of parallelization offered by contemporary CPUs occurs at instruction level in vector operations: registers can be up to 512-bit wide and, given some pre-requisites about memory alignment, it is possible to load up to 64 8-bit words in one operation and operate on them as a vector. For example, it is possible to add two 64 8-bit values with only one *add* instruction, when stored in such a vector configuration. When originally introduced in mid 1990's - as the Intel MMX instruction set extension - these features were available only as specialized low-level instructions, hence they required to manually write parts of code in assembly language. Nowadays, modern compilers implement several strategies of auto-vectorization, i.e. they are able to detect which part of the code are eligible for the use of vector instructions and automatically select them instead of the standard scalar primitives.

The current trend in CPU architecture is to follow the multi-core approach, increasing the number of cores and favoring a parallel approach in software design. In this context, a very interesting CPU architectures is the Intel MIC [174] whose current implementation is the Intel XeonPhi family of microprocessors: this is a particular derivation of standard x86-64 processors targeted to HPC infrastructures, that provides up to 72 cores and 4x hyper-threading engine for each core in a single CPU, hence they are able to execute up to 288 threads concurrently. Its instruction set provides support for 512-bit wide vector operations, so that 16 single or 8 double precision operations per cycle can be executed in parallel. They also feature an impressive 16GB of level 2 on-package cache memory that manages to hide latency in accessing data from the main RAM. One of the main feature that makes Intel MIC accessible to programmers is its support for common parallelization libraries such as MPI and OpenMP; moreover, unlike NVidia CUDA, MIC programming paradigm follows the standard C++11/14. Unfortunately, harnessing such impressive computing power requires an exceptional effort in software

design; moreover, few problems expose a level of parallelism that can be explicitly exploited for an efficient implementation over this family of CPUs. Nonetheless, in the next chapter we show how we designed a software able to fully benefit the features of the XeonPhi processors.

2.2.2 Heterogeneous computing: accelerators and GPUs

Another successful trend emerged in the latest years, which incidentally is another way to overcome the barriers that halted Moore's law trend, is represented by heterogeneous computing. Under this paradigm, the most resource demanding tasks are not executed on a general purpose CPU, but are off-loaded to specialized hardware which can execute these tasks more efficiently, in terms of computing time, memory usage or power consumption. This is not a new idea, as co-processors were available for early Intel processors since the introduction of the 8086 CPU, being the 8087 math co-processors an early example.

Currently, several families of hardware accelerators are available for scientific use: a few examples are re-programmable FPGA cards, Intel XeonPhi boards (XeonPhi as co-processors) and NEC Vector Accelerator Engine cards. All of them can be used to directly off-load a computation or cooperatively work with the main host CPU.

However, the most successful family of co-processors are Graphic Processing Units (GPU). As the name implies, GPUs were originally developed in the late 80's and early 90's to provide hardware acceleration for solving various tasks related to the graphical rendering problem. Silicon Graphics in particular pioneered the field, proposing in 1993 a hardware architecture comprising 353 independent processors called Reality Engine [175] and proposing one of the first open API for graphics programming, the OpenGL standard. However, Silicon Graphics workstations were notoriously expensive and mass diffusion of GPUs begun only in the early 2000's, as companies such as ATI and NVidia popularized their usage for video game applications in personal computers. Among these, in 2001 NVidia proposed the first consumer GPU featuring programmable graphic stages, hence clever programmers started to use GPU processors in a more general context, not strictly related to graphics: hence the term GP-GPU - General Purpose GPU - started to be used. Finally, in 2006 NVidia proposed the first official API for using GPUs in scientific application, Compute Unified Device Architecture (CUDA) and this paved the way for bringing parallel computation on a wide range of applications, also given that GPU processors are widespread across a large range of electronic devices, from smartphone to fully fledged workstation.

Current GPU processors feature a computing power that is one order of

magnitude higher than contemporary CPUs: their processor features hundreds, if not thousands of highly specialized computing cores working concurrently. A throughout description of GPU processors and the CUDA programming model is available in [176].

However, as often in parallel computation, GPU accelerators have several drawbacks that limit their field of applicability. First and foremost, not every problem is suitable for efficient execution on GPU processors: the problem must show a high level of fine grained parallelism, as execution efficiency increases if all the computing cores execute concurrently the same operation on different data (SIMD paradigm) and this does not hold true in several problems. Then, GPUs feature a limited amount of on-board memory that limits the size of treatable problems. Finally, GPU architecture - comprising the execution model and the memory hierarchy - is very complex; moreover, the CUDA programming paradigm itself is very hard to master. This has been partially solved in recent years, as several high-level libraries and frameworks have been proposed to leverage the difficulty of GPU programming: these libraries provides an interface for solving specific problems, hiding the underlying complexity of CUDA programming. A few examples are *cuBLAS*, *cuSPARSE* and *cuSOLVER* for dealing with linear algebra problems, *OpenCL* for image and video processing, *cuDNN* and *TensorFlow* for Deep Neural Network, and so on.

In chapter 4 we present how GPUs can be used in the context of precision medicine, proposing a GPU accelerated Deep Neural Network for predicting regulatory region activities; also, in chapter 5 we propose a GPU accelerated method for solving the Automated Protein Function problem; finally, in chapter 5 we show a more general use of GPUs, proposing a novel approach for solving the graph coloring problem.

2.2.3 Large scale computing

Multi-core, many-core and accelerator technologies provide hardware support for reducing computational times when facing small to medium size problems - more generally, problems that can fit on a single machine, given all its hardware constraints in terms of memory, disk space and computing power. However, large and massive problems can be tackled only on multiple machines that may or may not work cooperatively. In this context, Large Scaling Computing is a very broad term that encompasses a multitude of research and technological fields involved in writing, deploying, executing and analyzing techniques for solving a problem using a large number of computing resources.

Large computing systems are composed by a conspicuous number of inter-

connected computing machines (nodes). In a *cluster* or in a *grid*, computing nodes are concentrated and hosted in a single facility, and they are connected by a medium to fast network infrastructure; on the opposite, in a *distributed system*, computing nodes can be placed in different locations - even scattered around the globe - and are generally connected by a slow network.

The difference between a cluster and a grid lies in the interconnecting network, and this characteristics also influences their programming model and the scope of target applications. A cluster features a very fast and reliable networking infrastructure, so that each node can inter-exchange data with all the other nodes with a very small time overhead. Thanks to its fast interconnection scheme, nodes can also share their hardware resources: as an example, IBM General Parallel File System (GPFS) is a software abstraction layer providing clustered file system capabilities to a large parallel system; thanks to GPFS, the hard drives of all the computing nodes are shared across the network and perceived as a single big hard drive partition, instead of a set of small separated drives. Clusters are suitable for running tasks that require interactions between computing nodes, either for data sharing or for simple synchronization. One example can be finding the solution of a dynamic system using the Finite Elements Method (FEM) for partial derivative systems over a large domain: it is possible to divide the domain in subsets, assign each subdomain to a computing node and run the FEM solver on all the computing nodes concurrently. In this example, data exchange between nodes plays a critical role, as on every iteration of the FEM method each subdomain updates its state according to the state of each bordering subdomain, hence data exchange over the network must be exceptionally fast.

Programming clusters is fairly complicated as, in order to exploit the computational power offered by these systems, several details must be hand curated: just to name a few examples, programmers have to explicitly manage data and message passing between nodes, manage the granularity of the task in each node, provide each computing node a fair and balanced amount of work, design communication patterns so that computation and data exchange are overlapped to maximize efficiency, and so on. The matter is further complicated by the fact that each computing node may also have dedicated accelerators - i.e. a set of GPUs - hence leading to very complex hardware architectures.

On the opposite, grids are arrays of computing nodes interconnected by a regular network infrastructure. This characteristic makes grids suitable for running parallel tasks that require little to no data exchange or synchronization between nodes, as the limited network throughput represents the major bottleneck in computation. Grids are suitable for running the same operation over a set of independent data, for instance to apply a BLAST alignment on

a set of different sequences: as each sequence can be self contained on a dedicated file, one can choose to execute the alignment tool on a single machine, processing all the file sequentially, or use a grid to perform the alignment on multiple machine, assigning a subset of files to each node. For this reason, the programming model of a grid is not different than a local single machine.

The works presented in this document are targeted to both clusters and grids. In particular, in chapter 3 we present a ML solution that splits the workload across the node in a cluster; it explicitly orchestrates all the inter-communication and synchronization between processes. The works presented in Chapters 4 and 5 are instead targeted to a single workstation or a computing grid.

2.2.4 Software libraries and frameworks for large scaling computing

Most, if not all, programming languages that are in the top 20 TIOBE index chart provide some support for multi-threading or even multi-processing [177]. None of them, however provide direct or indirect support for multi-node programming. For this reason, inter-node communication and synchronization is managed through the use of third-party libraries or programming frameworks.

Message Passing Interface (MPI) [178] is the most used specification standard for the implementation of libraries for the message-passing parallel programming model. By using one of the MPI library implementation adhering to this standard, a programmer can design an application able to exchange data through messages between different processes. Also, the standard requires that processes exchanging messages may or may not be on a single local machine, and a software abstraction leverages the mechanism of accessing and managing the underlying network for correctly delivering messages across processes. The first draft for the MPI standard was proposed in late 1992 and over the years it has been subjected by a number of revisions, the latest being version 3.1 presented in 2015. Being a standardization, rather than a library, several implementations exist, such as OpenMPI [179] or IntelMPI [180].

The MPI standard is considered to be a very low level library: its high flexibility allows a programmer to fine tune every aspect of the communication process, but this comes at the cost of increased difficulty. For this reason, over the years several frameworks - some of them relying on MPI itself - have been proposed to hide all the complexity of message passing. *Google MapReduce* and *Apache Hadoop* [169] are two popular general purpose frameworks

for processing Big Data with a distributed model on a cluster. They are considered as multi-process extension of the map and reduce functions of functional programming languages, where a map function applies filtering and sorting on a dataset, and a reduce operation apply a summation. However, there is no guarantee that the entire process is fast, and using these framework only helps a programmer to access distributed hardware with less difficulties than writing an MPI application. More modern and efficient approaches are Apache Spark [168], Apache Flink [181], Apache Tez [182] and Google DataFlow [183], each of them targeted to specific usages (based on data size, pattern and the operations to be performed) and providing an Application Programming Interface which is less cumbersome than accessing the MPI standard. However, several comparative tests [184] showed that maximum performance is achieved when using the latter, hence the MPI standard is the de-facto choice for the works presented in this Thesis.

Most of the frameworks and libraries specific to Machine Learning and Bioinformatics do not provide any support for parallel programming and distribution of working tasks across a cluster. However, R and Python programming languages provides implementations for the MPI library. Also, a notable exception is TensorFlow library for Deep Neural Network[39], as it supports parallel and distributed training in a cluster, out fo the box. Also, the Microsoft Cognitive Toolkit [185] provides support for distributed and parallelized training.

2.3 HPC in Precision Medicine

As stated in Chapter 1, Big Data is characterized by tremendous data size which increases the computing power requirements for data processing and analysis [186]. This is not limited to Genomic or Biomedical dataset: as an example, if we consider the PANGAEA [187] collection, consisting in almost 400 thousands dataset for a total of more that 17 billions points, it is clear that High Performance Computing techniques and facilities are mandatory even for simple data analysis or quality assurance in most of the present day research fields. This problem is especially relevant in Precision and Genomic Medicine as the complexity and heterogeneity of data often requires advanced and exceptionally computing intensive data processing techniques [188]. The use of High Performance Computing hardware, for instance those presented in Section 2.2, is therefore mandatory for ensuring the feasibility and usability of complex analysis approaches. However, this is often not enough, as traditional data analysis tools have been proven not to scale well on large computing infrastructures and usually do not implicitly benefit the

added computational power provided by, for instance, hardware accelerators. For this reason, over the years, an increasing number of software solutions explicitly developed for exploiting HPC resources, have been proposed.

Several, if not all, research fields of Precision Medicine saw the development of HPC solutions for data analysis. In Pharmacology, for example, Molecular Dynamics and Molecular Docking are, respectively, the research field which simulates the dynamics of physical movements of atoms and molecules, and simulate the docking of molecules; they are both key research area in new drugs discovery and in recent years, the demand for MD studies by the pharmaceutical industries rapidly rose. For this reason, several HPC solutions have been proposed for this highly computing demanding task [189, 190, 191].

HPC also plays a central role in Physiology: as an example, over the years many HPC-oriented computational models of cardiac physiology were proposed, comprising models related to electrophysiology, bioelectricity, muscle physiology and fluidodynamics [192, 193]. Also, in neurophysiology few detailed HPC-oriented brain modeling simulation software have been proposed: for instance, [194, 195] propose two different methods for simulating whole brain neuron-neuron interaction over a GPU.

Possibly, Bioinformatics and Computational Biology are two of the most prominent research fields that benefit the most of the HPC approach. As a few examples, in recent years many GPU accelerated software solutions have been proposed for biochemical network simulation [196], whole-genome gene-gene interaction analysis [197, 198], epistasis detection [199, 200]; [201] reports how Intel MIC accelerators are used in the context of bioinformatics, in particular phylogenetics [202, 203] and epigenetics; [204] proposes a Xeon-Phi accelerated package for the whole-genome DNA methylation detection; as for large scale HPC supercomputer, [205] proposes a MPI-driven multi-node tool for phylogenetics, and [206] a MPI-driven multi sequence alignment tool.

Indeed, sequence alignment and analysis are two of the research fields in bioinformatics for which HPC made a great impact in the past few years. For several of the most used algorithms for sequence alignment, at least one parallel counterpart had been developed, targeting different architectures: the Smith and Waterman algorithm [207] had been accelerated for multicore [208], Intel MIC [209, 210, 211], MPI [212, 213], GPU [214, 215], GPU clusters [216] and many more. The Clustal family of multialignment algorithms [217] is another example: this algorithm is exceptionally expensive as it produces a score for every pair of sequences via a pairwise sequence alignment approach; recently an MPI-based multi-node implementation has been proposed in [218]. Basic Local Alignment Search Tool (BLAST) [219] is another

popular algorithm for sequence comparison and local alignment, and several parallel versions have been developed for various platforms [220, 221, 222].

As for sequence analysis, the situation is a bit different, as few Bioinformatics software were expressively developed with HPC goals. Other than the above mentioned packages, we notice that even the popular *SAMTools* [223] used for post-processing DNA readings, is at best multi-threaded. Several processing pipeline relies on these tools and each step is inherently sequential. Also, many packages commonly used in Bioinformatics are written in the R programming language, whose interpreter is single-threaded and not performance oriented. However, a few sparse exceptions exist: *Hail* is an open-source library for scalable genomic data exploration based on Spark; [224] proposes a parallel pipeline for identifying genomic variants from whole-exome sequencing data; [225] proposes a GPU-accelerated analysis method for Shotgun Metagenomics; [226] proposes a MPI distributed algorithm for similarity clustering of nucleotide sequences; *CoVaCs* [227] is a parallel system for genotyping and variant annotation of resequencing data.

Unsurprisingly, Machine Learning resources specifically developed for HPC-oriented Bioinformatics uses are quite scarce. A few methods are listed in [228], but unfortunately, most of them are more suitable for distributed architectures, rather than supercomputers; also, many of the listed ML tools are generic ML algorithms, hence not properly developed for the key challenges of Bioinformatics.

Conversely, several HPC ML tools are available, starting from parallel implementations of basic classifiers such as Random Forest [229], Support Vector Machines for both Intel MIC [230] and GPU [231], and so on, with more few examples in [232]. Also, HPC -oriented domain -specific ML applications have been proposed: for instance, [233] describes an RF -based parallel algorithm for image segmentation, and [234] shows a parallel MPI -based SVM method for image classification.

Our work aims towards the development of new HPC -oriented ML algorithms with specific applications in several research fields of Bioinformatics by presenting, in the following Chapters, three applications in this context to fill this gap.

Chapter 3

A HPC hyper-ensemble ML system for the genome-wide prediction of pathogenic variants in the non-coding DNA

Several prediction problems in Computational Biology and Genomic Medicine are characterized by both big data as well as a high imbalance between examples to be learned, whereby positive examples can represent a tiny minority with respect to negative examples. For instance, deleterious or pathogenic variants are overwhelmed by the sea of neutral variants in the non-coding regions of the genome: as a consequence the prediction of deleterious variants is a very challenging highly imbalanced classification problem, and classical prediction tools fail to detect the rare pathogenic examples among the huge amount of neutral variants or undergo severe restrictions in managing big genomic data.

To overcome these limitations we propose *parSMURF*, a method that adopts a hyper-ensemble approach and oversampling and undersampling techniques to deal with imbalanced data, and parallel computational techniques to both manage big genomic data and significantly speed-up the computation. The synergy between Bayesian optimization techniques and the parallel nature of *parSMURF* enables efficient and user-friendly automatic tuning of the hyper-parameters of the algorithm, and allows specific learning problems in Genomic Medicine to be easily fit. Moreover, by using MPI parallel and machine learning ensemble techniques, *parSMURF* can manage big data by partitioning them across the nodes of a High Performance Computing cluster.

Results with synthetic data and with single nucleotide variants associated

with Mendelian diseases and with GWAS hits in the non-coding regions of the human genome, involving millions of examples, show that *parSMURF* achieves state-of-the-art results and a speed-up of 80× with respect to the sequential version.

The C++ OpenMP multi-core version tailored to a single workstation and the C++ MPI/OpenMP hybrid multi-core and multi-node *parSMURF* version tailored to a High Performance Computing cluster are both available from github: `https://github.com/AnacletoLAB/parSMURF`

This Chapter has been published in [235].

3.1 *parSMURF*^{1} and *parSMURF*^{n}

3.1.1 Background

High throughput bio-technologies, and the development of Artificial Intelligence methods and techniques has opened up new research avenues in the context of the Genomic and Personalized Medicine [236, 237]. In particular Machine Learning [238], whole-genome sequencing (WGS) technologies [239, 240], and large population genome sequencing projects [241, 242] play a central role for the detection of rare and common variants associated with genetic diseases and cancer [73, 74].

In this context, while disease-associated variants falling in the protein-coding regions of the genome have been largely studied [243, 244, 245], this is not the case for disease-associated variants located in the non-coding regions of the genome, where our understanding of their impact on cis and trans-regulation is largely incomplete. Nevertheless, several studies ended up that most of the potential pathogenic variants lie in the non-coding regions of the human genome [246].

Driven by the aforementioned motivations many efforts have been devoted in recent years by the scientific community to develop reliable tools for the identification and prioritization of "relevant" non-coding genetic variants. CADD is one of the first machine learning-based method applied for this purpose on a genome-wide scale [98]. By combining different annotations into a single measure for each variant using firstly an ensemble of support vector machines and in the current version a fast and efficient logistic regression classifier, CADD likely represents the most used and well-known tool to predict deleterious variants [99].

Starting from this work other machine learning-based methods for the detection of deleterious or pathogenic variants have been proposed, ranging from multiple kernel learning techniques [102], to deep neural networks [100,

105], multiple kernel learning integrative approaches [102], unsupervised learning techniques to deal with the scarcity of available annotations [104], and linear models for functional genomic data combined with probabilistic models of molecular evolution [103]. Other approaches predicted the effect of regulatory variation directly from sequence using gkm-SVM [107] or deep learning techniques [247]. More details are covered in two recent reviews on machine learning methods for the prediction of disease risk in non-coding regions of the human genome [248, 249].

All these tools are faced with relevant challenges related to the rarity of non-coding pathogenic mutations. Indeed neutral variants largely outnumber the pathogenic ones. As a consequence the resulting classification problem is largely unbalanced toward the majority class and in this setting it is well-known that imbalance-unaware machine learning methods fail to detect examples of the minority class (i.e pathogenic variants) [250]. Recently several methods showed that by adopting imbalance-aware techniques we can significantly improve predictions of pathogenic variants in non-coding regions. The first one (GWAVA) applied a modified random forest [251], where its decision trees are trained on artificially balanced data, thus reducing the imbalance of the data [101]. A second one (NCBoost) used gradient tree boosting learning machines with partially balanced data, achieving very competitive results in the prioritization of pathogenic Mendelian variants, even if the comparison with the other state-of-the-art methods have been performed without retraining them, but using only their pre-computed scores [110]. The unbalancing issue has been fully addressed by ReMM [108] and *hyperSMURF* [109], through the application of subsampling techniques to the "negative" neutral variants, and oversampling algorithms to the set of "positive" pathogenic variants. Moreover a large coverage of the training data and an improvement of the accuracy and the diversity of the base learners is obtained through a partition of the training set and a hyper-ensemble approach, i.e. an ensemble of random forests which in turn are ensembles of decision trees. *hyperSMURF* achieved excellent results in the detection of pathogenic variants in the non-coding DNA, showing that imbalance-aware techniques play a central role to improve predictions of machine learning methods in this challenging task.

Nevertheless these imbalance-aware methods have been usually implemented with no or very limited use of parallel computation techniques, thus making problematic their application to the analysis of big genomic data. Furthermore, the *hyperSMURF* method is computationally intensive and characterized by a large number of learning parameters that need to be tuned to ensure optimal performance, thus requiring prohibitive computing costs, especially with big genomic data.

To address the aforementioned limitations, we propose *parSMURF*, a novel parallel approach based on *hyperSMURF*. While other methods suitable for the assessment of the impact of variants located in protein-coding regions are able to run in parallel environments [252], this is not true for the assessment of non-coding variants. The main goal in the design and development of *parSMURF* is to make available to the scientific community a general and flexible tool for genomic prediction problems characterized by big and/or highly imbalanced data, while ensuring state-of-the-art prediction performance. The high computational burden resulting by the proper tuning and selection of the learning (hyper)-parameters is addressed through a scalable and parallel learning algorithm that leverages different levels of parallelization, and a Bayesian optimizer for their automatic and efficient tuning.

Therefore, in this setion we present the *parSMURF* algorithm, its relationships with its sequential version *hyperSMURF*, and its two different implementations respectively for multi-core workstations and for a highly parallel High Performance Computing cluster. In the Results section experiments with big synthetic and actual genomic data show that *parSMURF* scales nicely with big data and significantly improves the speed-up of the computation. Finally experiments with Mendelian data and GWAS hits at whole-genome level show that *parSMURF* significantly improves over its sequential couterpart *hyperSMURF*, by exploiting its multiple levels of parallelism and the automatic tuning of its learning hyper-parameters through a grid search or a Bayesian optimization method. *parSMURF*^{1} multi-thread and hybrid multi-thread and multi-process MPI C++ *parSMURF*^{n} implementations are well-documented and freely available for academic and research purposes.

3.1.2 Methods

Parallel SMote Undersampled Random Forest (*parSMURF*) is a fast, parallel and highly scalable algorithm designed to detect deleterious and pathogenic variants in the human genome. The method is able to automatically tune its learning parameters even with large data sets, and to nicely scale with big data.

Starting from the presentation of the characteristics and limitations of *hyperSMURF* [109], in this section we introduce the parallel algorithm *parSMURF* and its two variants named *multi-core* parSMURF (*parSMURF*^{1}) and *multi-node* parSMURF (*parSMURF*^{n}). The first one is suitable for the execution on a single workstation that features one or more multi-core processors, while the second one is designed for a High Performance Computing cluster, where the computation is distributed across several interconnected nodes. Although

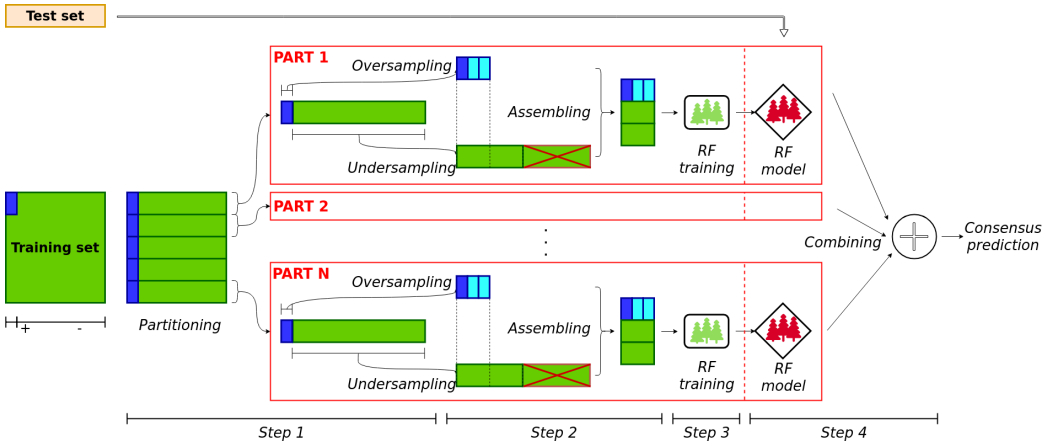


Figure 3.1: High-level scheme of *hyperSMURF*. Step (1): partitioning of the training set (the minority/positive class is represented in blue, while the majority/negative class is in green). Step (2): application of oversampling and undersampling approaches, and assembling of the training set. Step (3): training of the RF models. Step (4): Testing and aggregation of predictions outcomes.

developed for different hardware architectures, they both share the same core parallelization concepts and the same chain of operations performed on each parallel component of the algorithm. Finally, we discuss the computational algorithms proposed to automatically learn and tune the *parSMURF* hyper-parameters in order to properly fit the model to the analyzed genomic data.

From *hyperSMURF* to *parSMURF*

hyperSMURF is a supervised machine learning algorithm specifically designed to detect deleterious variants where variants associated with diseases are several order of magnitude lesser than the neutral genetic variations. *hyperSMURF* tackles the imbalance of the data using three learning strategies:

- balancing of the training data by oversampling the minority class and undersampling the majority class;
- improving data coverage through ensembling techniques;
- enhancing the diversity and accuracy of the base learners through hyper-ensembling techniques.

The high-level logical steps of the *hyperSMURF* algorithm are summarized in Figure 3.1. At step (1) *hyperSMURF* creates several sets of training

data by using all the available examples of the minority (positive) class and partitioning the set of the majority (negative) class: as a result each set includes all the positive examples and a subset of the majority (negative) class. From this point on, each training set is processed independently. In step (2), examples of the minority class are oversampled through the SMOTE algorithm [253] while examples of the majority class are undersampled according to an uniform distribution. Each training set is now formed by a comparable number of positive and negative examples and it can be used in step (3) to train the random forest. This process is applied to all the parts of the partition of the original training set, thus generating an ensemble of random forest models. At step (4) all the predictions separately computed by each trained model are finally combined to obtain the "consensus" prediction of the hyper-ensemble.

The behavior of the algorithm strongly depends on the learning hyper-parameters, reported in Table 3.1, which deeply influence the *hyperSMURF* performance, as shown in [254], and fine tuning of the learning parameters can dramatically improve prediction performance. Since *hyperSMURF* is an ensemble of random forests which in turn are ensembles of decision trees, its sequential implementation undergoes a high execution time, especially on large datasets, thus limiting a broad exploration of the hyper-parameter space. Moreover *hyperSMURF* cannot be easily applied to big data, due to its time and space complexity issues.

Parameters	**Description**
`nParts`	Number of parts of the partition
`fp`	Multiplicative factor for oversampling the minority class. For instance with $fp = 2$ two novel examples are synthesized for each positive example of the original data set, according to the SMOTE algorithm.
`ratio`	Ratio for the undersampling of the majority class. For instance, $ratio = 2$ sets the number of negatives as twice the total number of original and oversampled positive examples.
`k`	Number of the nearest neighbors of the SMOTE algorithm
`nTrees`	Number of trees included in each random forest
`mtry`	Number of features to be randomly selected at each node of the decision trees included in the random forest

Table 3.1: *hyperSMURF* learning hyper-parameters

Nevertheless, looking at Figure 3.1, we can observe that *hyperSMURF* is

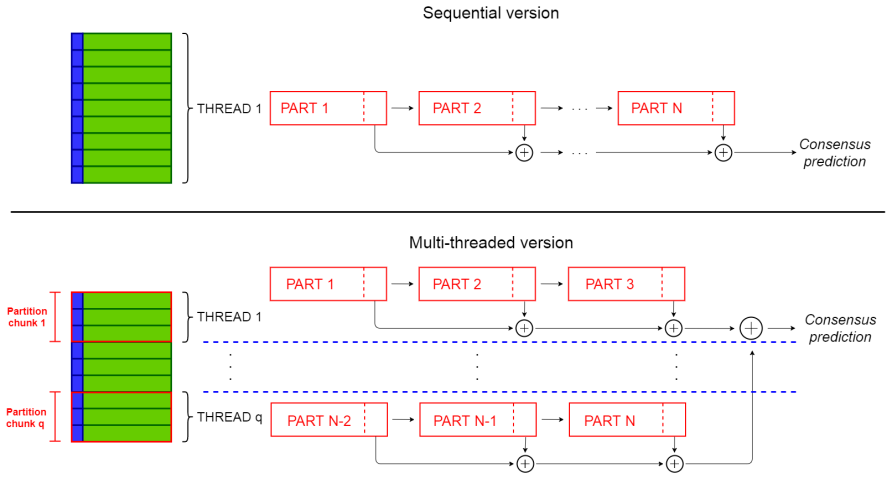


Figure 3.2: Comparison between the sequential *hyperSMURF* (top) and multi-core *parSMURF*[1] (bottom) execution schemes.

characterized by the following features:

1. the same operations (over- and under-sampling, data merging, training and model generation and prediction evaluation) are performed over different data belonging to different partitions;

2. the operations performed over different data are independent, i.e. there is no interaction between the computation of two different partitions;

3. the algorithm does not require any explicit synchronization during the elaboration of two or more partitions.

Putting together these observations, we can redesign *hyperSMURF* leveraging its intrinsic parallel nature and using state-of-the-art parallel computation techniques. The resulting newly proposed *parSMURF* algorithm is schematically summarized in Algorithm 1. The parallelization is performed by grouping parts of the partition in *chunks* (see also Figure 3.2). The *parSMURF* parameter q (number of chunks) determines at high level the parallelization of the algorithm, i.e. how many chunks can be processed in parallel.

During training, the main activities of the *parSMURF* algorithm are executed in parallel for each chunk (outer `for` loop in Algorithm 1). A further level of parallelism can be realized through the inner `for` loop where each part $\mathcal{N}'$ of the chunk $\mathcal{C}_i$ undergoes a parallel execution. Note however that "parallel" in the inner `for` loop is in brackets to highlight that this second level

Algorithm 1 *parSMURF* algorithm (training)

Input:
$\mathcal{P}$: positive examples set
$\mathcal{N}$: negative examples set
$n, fp, ratio, k, nTrees, mtry$: parameters described in Table 3.1
q: number of partition chunks
Output:
$M = \{m_1, .., m_n\}$: set of trained RF models

$\{\mathcal{N}_1, .., \mathcal{N}_n\} \leftarrow$ partitioning($\mathcal{N}$, n)
$\{\mathcal{C}_1, .., \mathcal{C}_q\} \leftarrow$ chunkGroups($\{\mathcal{N}_1, .., \mathcal{N}_n\}$, q)
$idx \leftarrow \{1, .., q\}$
$j \leftarrow 0$
$M \leftarrow \emptyset$
for all $i \in idx$ **parallel do**
  for $\mathcal{N}' \in \mathcal{C}_i$ (**parallel**) **do**
    $j \leftarrow j + 1$
    $\mathcal{P}' \leftarrow SMOTE(\mathcal{P}, fp, k)$
    $\mathcal{N}'' \leftarrow undersample(\mathcal{N}', ratio)$
    $T \leftarrow \mathcal{P} \cup \mathcal{P}' \cup \mathcal{N}''$
    $m_j \leftarrow RF_{train}(T, nTrees, mtry)$
    $M \leftarrow M \cup m_j$
  end for
end for
return $M = \{m_1, .., m_n\}$

of parallelization can or cannot be implemented, according to the complexity of the problem and the available underlying computational architecture.

Algorithm 2 also shows that hyper-ensemble predictions conducted during testing can be easily performed through parallel computation: each model can be tested independently over the same test set and the consensus prediction is computed by averaging the ensemble output.

Multi-core *parSMURF*: *parSMURF*[1]

The idea on which multi-core *parSMURF* builds upon is that all operations performed on the different parts of the partition can be assigned to multiple core/threads and processed in parallel. Namely, given q threads, the data parts $N_1, \ldots, N_n$ are equally distributed among threads so that thread

Algorithm 2 *parSMURF* algorithm (testing)

Input:
$M = \{m_1, .., m_n\}$: set of trained RF models
$\mathcal{T}$: test set
Output:
HS: prediction score for each example t in test set $\mathcal{T}$

$idx \leftarrow \{1, .., n\}$
for all $i \in idx$ **parallel do**
  for all $t \in \mathcal{T}$ **do**
    $P_i(t) \leftarrow \mathrm{P}(\text{t is positive } |m_i)$
  end for
end for
for all $t \in \mathcal{T}$ **do**
  $HS(t) \leftarrow \frac{1}{n} \sum_{i=1}^{n} P_i(t)$
end for

i receives a subset (chunk) C_i of parts, and processes its assigned parts in sequence. Since each partition chunk is assigned to its own thread, chunk processing is performed in parallel with architectures featuring multiple processing cores.

In Figure 3.2 (top) a scheme of the execution of the sequential *hyper-SMURF* is shown: each partition is processed sequentially and the output predictions are accumulated as the computation goes on. On the contrary, with *parSMURF*[1] (Figure 3.2, bottom), chunks of partitions are assigned to different execution threads and are processed in parallel. To avoid data races, each thread accumulates partial results, and then the master thread collects them once the computation of each thread is ended. Moreover, each thread keeps only a local copy of the subset of the data which is strictly required for its computation; this minimizes memory consumption and, at the same time, does not impose any need for synchronization between concurrent threads.

This scheme is expected to show a remarkable speedup with the increase of processing cores and the available local memory of the system. Since parallelization occurs at "partition chunk" level, instances of *parSMURF*[1] with a reduced partition size benefits only partially of a multicore execution. On the other side, partitions characterized by a very high number of parts can theoretically scale well with the number of processing cores but, unfortunately, current processors have constraints in the number of available cores.

Moreover, big data computation may exceed the storage capacity of a single workstation, thus making the application of *parSMURF*[1] in this experimental context problematic.

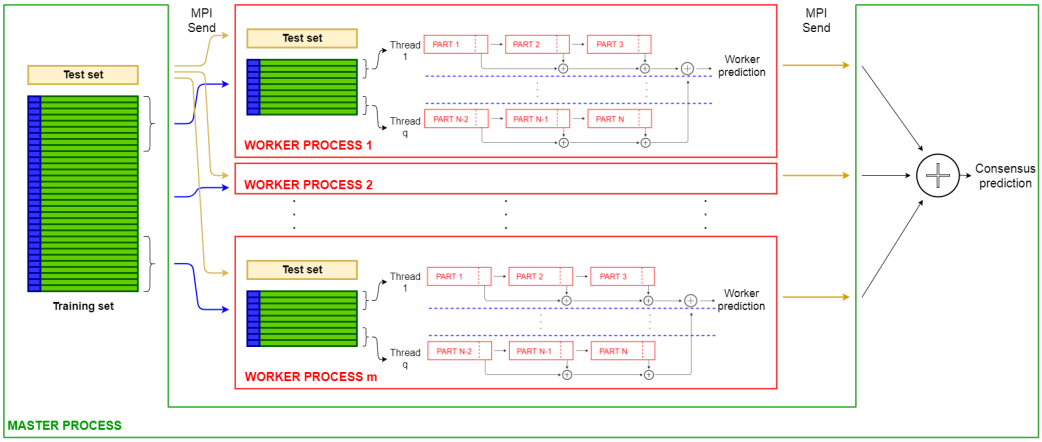


Figure 3.3: High-level scheme of the multi-node *parSMURF*^{n} implementation.

Multi-node *parSMURF*: *parSMURF*^{n}

This version of *parSMURF* has been designed to process big data, to both improve speed-up and make feasible computations that exceed the storage capacity of a single workstation. Moreover *parSMURF*^{n} allows the fine tuning of the model parameters even when big data should be analyzed.

parSMURF^n architecture

As for the multi-core version, *parSMURF*^{n} exploits parallelization at partition level, but also introduces a second level of parallelization: the higher level is performed through the computing nodes of a cluster, i.e. a set of computing machines interconnected by a very fast networking infrastructure; the lower level is realized through multi-threading at single node level by exploiting the multi-core architecture of each single node of the cluster. In this novel scheme, each node receives a partition chunk, which is processed in parallel with the other chunks assigned to the remaining nodes. Chunks in turn are further partitioned in sub-chunks, distributed among the computing cores available at the local node. Finally an optional third level of parallelization is available by assigning multiple threads to the random forests which process the different parts of the partition (recall that a random forest is in turn an ensemble of decision trees).

The higher level of parallelization leverages the MPI programming paradigm and standard [178] to transfer information among nodes. This programming paradigm requires that several instances of the same program are executed concurrently as different processes (*MPI processes*) on different nodes interconnected by a network. Being different instances of the same program, each MPI process has its own memory space, therefore intercommunication, i.e. data exchange between processes, occurs explicitly by invoking the corresponding actions, as required by the MPI standard.

parSMURF^n adopts a master–slave setting, with a master process coordinating a set of MPI slave processes, also called *worker processes*, which in turn manage the partition computation. Master and worker roles are described below:

- the *master process* is responsible for processing the command line parameters, loading data in its memory space, generating partition and chunks, sending the proper subset of data to each worker process (including the test set and the proper fraction of the training set) and finally collecting and averaging their output predictions.

- each *worker process* realizes the computation on the assigned chunk of partitions, generates sub–chunks of its own chunk and processes them through multi-threading - i.e. distributes the computation of the sub–chunks over the available computing threads - and sends the output predictions back to the master process.

We point out that in principle *parSMURF*^{n} can be executed also on a single machine, where multiple copies of the same program are processed by the available cores, but in this case it undergoes the same limitations of *parSMURF*^{1}. Figure 3.3 provides a high level scheme of the execution of *parSMURF*^{n}.

parSMURF^n intercommunication

Figure 3.4 shows a schematic view of the intercommunication between *parSMURF* MPI processes.

The computation in the worker processes is performed as in the multicore version of *parSMURF*, except for a slight difference in the subsampling of the majority class, since this operation is no longer executed by the worker processes but by the master instead. Indeed, by observing that subsampling requires some examples to be discarded, there is no need of sending to the worker processes an entire part of the partition, but only the selected subset of examples. This design choice minimizes the amount of data to be sent to

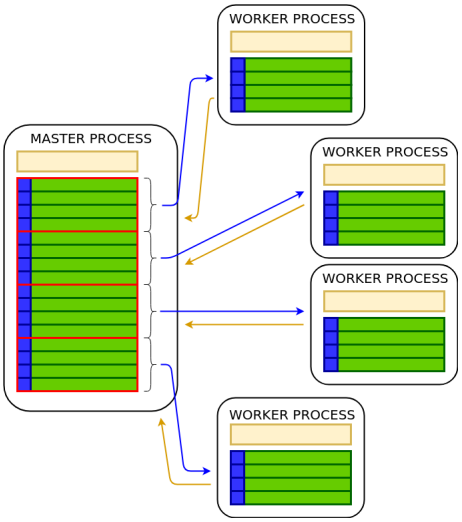


Figure 3.4: High-level intercommunication scheme between MPI processes in multi-node *parSMURF*^{n}. Blue arrows represent data flows from the master process to worker processes (different chunks of partitions and the same test set) and yellow arrows represent data flows from the worker processes to the master (output predictions).

a worker process, since for each partition only the positive samples (that are going to be oversampled in the worker process) and the already undersampled negative examples are sent to the worker processes.

In an ideal parallel application, computing nodes should never interact, since every data exchange creates latencies that affect the overall *occupancy* - i.e. the ratio between the amount of time a computing node is processing data and the total execution time. However, in real applications this rarely happens, and data have to be exchanged between processes. As a general rule, communication should be minimal in number and maximal in size, since the following equality holds:

$$t_{tot} = t_{start} + d \times t_{trn}$$

where t_{tot} is the total time for the data send, d is the amount of data in bytes to be transferred, t_{trn} is the time required to transfer one byte of data and t_{start} is the time required by the interconnecting networking system for beginning a communication between nodes. t_{start} is constant, therefore transferring data as a big chunk is generally faster than several smaller ones, since t_{start} penalty is paid only once in the former case.

However, in real world MPI parallel applications, the main objective is to parallelize computation to speed-up execution, and maximum efficiency is

achieved by overlapping data transmission and computation. This is easier to obtain when data is *streamed*, i.e. sent in small chunks which are consumed as soon as they reach the receiver MPI process: in this way we can minimize the inactivity time of a node, waiting for data to be received.

Maximizing *parSMURF*^{n} performance

For maximizing performances *parSMURF*^{n} adopts the following strategies to find the optimal balance between the size and number of data transmissions:

- *maximize occupancy*,
- *reduce the number of data send* or *broadcast*,
- *minimize latency*.

To *maximize occupancy*, the master process does not send the entire chunk of partitions to each worker process in a big data send; instead, parts are sent one by one. This choice is ideal in the context of multithreading in worker processes: supposedly, given a partition with n parts and a number x of computing threads assigned to a working process, the master at first sends to each worker process x parts of its assigned chunk. When a worker thread finishes the computation of a part, another one is sent by the master for processing. This process goes on until the chunk is exhausted.

To *reduce the number of data send* or *broadcast* - i.e. one MPI process sending the same data to all the other processes - for each part, the master process assembles an array having all the relevant data required for the computation, i.e. the positive and negative examples (already subsampled, as stated earlier) and the parameters needed for the computation. Hence with just one MPI send operation, a part of the partition with its parameters is transferred to the worker process. Also, partial results of the predictions are locally accumulated inside each worker and sent to the master once the jobs for the assigned chunk are finished.

To *minimize latency*, the assembly of the data to be sent is multithreaded in the master process. In instances characterized by relatively small datasets and a high number of parts in the partition, it may happen that the master could not prepare and send data fastly enough to keep all the worker processes busy. For instance, a thread in a worker process may finish the computation for a part before data for the next one arrive, leaving the thread or the entire process inactive. This has been solved by spawning a number of threads in the master process equal to the number of worker processes the user has

requested, each one assigned for preparing and sending the data to the corresponding worker. However, since memory usage in the master process can be greatly affected, an option is provided for disabling multithreading in the master. In this case, only one thread takes care of this task and parts are sent in round robin fashion to the working processes: this has been experimentally proven to be effective for those instances that require a particularly high memory usage.

Hyper-parameters tuning

As in most ML methods, the accuracy of the predictions of the *parSMURF* models are directly related to the set S of hyper-parameters that control its learning behaviour. Hence, to maximize the usefulness of the learning approach, the value of each hyper-parameter of the set S must be chosen appropriately. In *parSMURF* the hyper-parameter set is composed by the 6-tuple of parameters reported in Table 3.1. Each parameter is discretized and constrained between a maximum and minimum value, hence the hyper-parameter space $\mathcal{H}$ is a discrete six-dimensional hypercube. The validation procedure for the evaluation of each $h \in \mathcal{H}$ is the internal cross-validation process, and the objective function (performance metrics) which has to be maximized is the Area Under the Precision Recall Curve (AUPRC).

parSMURF features two modes for automatically finding the hyper-parameters combination $h_0 \in \mathcal{H}$ that maximizes the model accuracy (parameter auto-tuning). The first strategy is a grid search over H, where each $h \in \mathcal{H}$ is evaluated by internal cross-validation. This strategy is generally capable of finding values close to the best hyper-parameters combination, but it is very computationally intensive and suffers from the curse of dimensionality. The second strategy is based on a Bayesian Optimizer (BO) which iteratively builds a probabilistic model of the objective function $f : \mathcal{H} \to \mathbb{R}^+$ (in *parSMURF*, the AUPRC) by evaluating a promising hyper-parameter combination at each iteration and stopping when a global maximum of f is obtained. This procedure is less computationally intensive than the grid search and may also outperform the latter in terms of prediction effectiveness. The Bayesian optimizer is based on [255] and its implementation "Spearmint-lite" [256] is included in the *parSMURF* package.

The whole procedure is summarized in Algorithm 3. Briefly, *parSMURF* provides the automatic tuning of the hyper-parameters in a context of internal n-fold cross-validation, and the Bayesian Optimizer (BO) is invoked in the while loop. At each iteration, a new hyper-parameter combination $h \in \mathcal{H}$ is generated by taking into account all the previously evaluated h. A new model is then trained and tested in the internal cross-validation procedure

Algorithm 3 Automatic hyper-parameters tuning in *parSMURF* featuring Bayesian Optimization.

Input:
$\mathcal{T}$: $\mathcal{P} \cup \mathcal{N}$ (data for training and validation)
$\mathcal{H}$: Hyper-parameter space
n: number of CV folds
$maxIter$: maximum number of iterations
ε: Bayesian Optimization (BO) error tolerance
Output:
H': set of best combination (one of reach fold) of hyper-parameters

$\{\mathcal{T}_1, .., \mathcal{T}_n\} \leftarrow$ foldSubdivision($\mathcal{T}$, n)
for $i \in \{1, .., n\}$ **do**
  $TestSet \leftarrow \mathcal{T}_i$
  $TrainingSet \leftarrow \bigcup_{j \neq i} \mathcal{T}_j$
  $\{\mathcal{T}'_1, .., \mathcal{T}'_{n-1}\} \leftarrow$ foldSubdivision($TrainingSet$, $n-1$)
  $iter \leftarrow 1$
  $H \leftarrow \emptyset$
  while $(error > \varepsilon) \wedge (iter < maxiter)$ **do**
    $h \leftarrow$ BO_generate(H)
    for $k \in \{1, .., n-1\}$ **do**
      $ValidationSet \leftarrow \mathcal{T}'_k$
      $TrainingSet' \leftarrow \bigcup_{l \neq k} \mathcal{T}'_l$
      $model' \leftarrow$ parSMURF_train($TrainingSet', h$)
      $predictions'_k \leftarrow$ parSMURF_test($model', ValidationSet$)
    end for
    $predictions' \leftarrow \bigcup_k predictions'_k$
    $eval \leftarrow$ AUPRC($predictions'$)
    $H \leftarrow H \cup (h, eval)$
    $error \leftarrow$ BO_errorEval(H)
    $iter \leftarrow iter + 1$
  end while
  $h_i \leftarrow argmax_{h \in H} \{(h, eval)\}$
  $model \leftarrow$ parSMURF_train($TrainingSet, h_i$)
  $predictions \leftarrow$ parSMURF_test($model, TestSet$)
end for
$H' \leftarrow \bigcup_k h_k$

by using the newly generated h. The quality of the prediction is evaluated by means of AUPRC, and the tuple $(h, eval)$ is submitted back to the Bayesian Optimizer for the generation of the next h. The while loop ends when the Bayesian Optimizer finds a probable global maximum (no further improvement in the error evaluation) or when the maximum number of iterations is reached. Grid search works in a similar way, but all $h \in \mathcal{H}$ are exhaustively tested in the internal cross-validation phase.

3.1.3 Results and Discussion

We applied *parSMURF* to both synthetic and real genomic data to show that the proposed method is able to:

- scale nicely with big data;
- auto-tune its learning parameters to optimally fit the prediction problem under study;
- improve *hyperSMURF* as well as other state-of-the-art methods in the prediction of pathogenic variants in Mendelian diseases and of regulatory GWAS hits in the GWAS Catalog.

All the experiments have been performed on the Cineca Marconi Supercomputing system [257], specifically using its Lenovo NeXtScale architechture, with 720 nodes, each one having 128 GBytes of RAM and 2×18-cores Intel Xeon E5-2697 v4 (Broadwell) CPUs at 2.30 GHz. The interconnecting infrastructure is a Intel Omnipath featuring 100 Gb/s of networking speed and a fat-tree $2:1$ topology.

Datasets

Genomic data are highly imbalanced toward the majority class, since the SNVs annotated as pathogenic represent a tiny minority of the overall genetic variation. Synthetic data have also been generated to obtain a high imbalance between positive and negative examples, in order to simulate the imbalance that characterizes several types of genomic data.

Synthetic data have been randomly generated using a spheric gaussian distribution for each of the 30 features. Among them only 4 are informative in the sense that the means of positive and negative examples are different, while all the other features share the same mean and standard deviation with both positive and negative examples. Synthetic data, as well as the R code for their generation are available from the GitHub repository [258].

As an example of application of *parSMURF* to real genomic data, we used the dataset constructed in [108] to detect Single Nucleotide Variants (SNVs) in regulatory non-coding regions of the human genome associated with Mendelian diseases. The 406 positive examples are manually curated and include mutations located in genomic regulatory regions such as promoters, enhancers and 5' and 3' UTR. Neutral (negative) examples include more than 14 millions of SNVs in the non-coding regions of the reference human genome differing, according to high confidence alignment regions, from the ancestral primate genome sequence inferred on the basis of the Ensembl Enredo-Pechan-Ortheus whole-genome alignments of six primate species [259], and not including variants present in the most recent 1000 Genomes data [241] with frequency higher than 5% to remove variants that had not been exposed for sufficiently long time to natural selection. The imbalance between positive (mutations responsible for a Mendelian disease) and negative SNVs amounts to about 1:36,000. The 26 features associated to each SNV are genomic attributes ranging from G/C content, population-based features, to conservation scores, transcription and regulation annotations (for more details, see [108]).

We finally analyzed GWAS (Genome Wide Association Studies) data to detect 2115 regulatory GWAS hits downloaded from the National Human Genome Research Institute (NHGRI) GWAS catalog [260], and a set of negatives obtained by randomly sampling 1/10 of the negative examples of the Mendelian data set, according to the same experimental set-up described in [109], thus resulting in an imbalance between negative and positive examples of about $1:700$. We predicted chromatin effect features directly from the DNA sequence using DeepSEA convolutional networks [105]: in this way we obtained 1842 features for each SNV, as described in [109], and we used that features to train *parSMURF*.

Table 3.2 summarizes the main characteristics of both the synthetic and genomic data used in our experiments.

parSMURF scales nicely with synthetic and genomic data

Experiments reported in this section follow the classic experimental setup for the evaluation of the performances of parallel algorithms [261]. In particular, since our executions employ multiple computer processors (CPU) concurrently, we use speedup and efficiency to analyze the algorithm performances by measuring both the sequential and parallel execution times.

By denoting with T_s the run-time of the sequential algorithm and with T_p the run-time of the parallel algorithm executed on P processors, the speedup

Table 3.2: Summary of the datasets used in the experiments. Datasets are highly imbalanced towards the negative class.

Name	Number of samples	Number of features	Number of positive samples	Imbalance ratio
synth_1	10^6	30	400	1 : 2500
synth_2	10^7	30	400	1 : 25000
synth_3	5×10^7	30	1000	1 : 50000
Mendelian	14.755.605	26	406	1 : 36300
GWAS	1.477.630	1842	2115	1 : 700

and efficiency are defined respectively as:

$$S = \frac{T_s}{T_p} \qquad \text{and} \qquad E = \frac{T_s}{T_p \times P}.$$

Speed-up and efficiency analysis with synthetic data

For every synthetic dataset, we run *parSMURF*^{1} and *parSMURF*^{n} several times by varying the number of threads (for both the multi-core and multi-node versions) and the number of MPI worker processes assigned to the task (for the multi-node version only). More precisely the number of threads *n.thr* varied in $n.thr \in \{1, 2, 4\}$ for synth_1 and synth_2 datasets, while for synth_3 $n.thr \in \{1, 2, 4, 8, 16, 20\}$. Moreover we considered a number of MPI processes *n.proc* in the range $n.proc \in \{1, 2, 4, 8\}$ for synth_1 and synth_2, and $n.proc \in \{1, 2, 4, 6, 8, 10\}$ for synth_3.

For each run we collected the execution time and evaluated the speed-up and efficiency: the T_s sequential time of formulas (1) and (2) has been obtained by running *parSMURF*^{1} with 1 thread, hence obtaining a pure sequential run.

All experiments were executed using a 10-fold cross validation setting. The learning hyper-parameters employed in each experiment are the following:

- synth_1: $nParts = 128$, $fp = 1$, $ratio = 1$, $k = 5$, $nTrees = 128$, $mTry = 5$;
- synth_2: $nParts = 64$, $fp = 1$, $ratio = 1$, $k = 5$, $nTrees = 32$, $mTry = 5$;
- synth_3: $nParts = 128$, $fp = 1$, $ratio = 5$, $k = 5$, $nTrees = 10$, $mTry = 5$

For each synthetic data set we run experiments considering all the different numbers of threads *n.thr* for $parSMURF^1$, while for $parSMURF^n$ we run different hyper-ensembles considering all the possible combinations of *n.thr* and *n.proc*.

Figure 3.5 reports the results for the batch of executions with the synth_1 and synth_2 datasets. Results are grouped by the number of MPI working processes (each line represents three runs obtained by keeping the number of MPI processes fixed and by varying the number of threads per process).

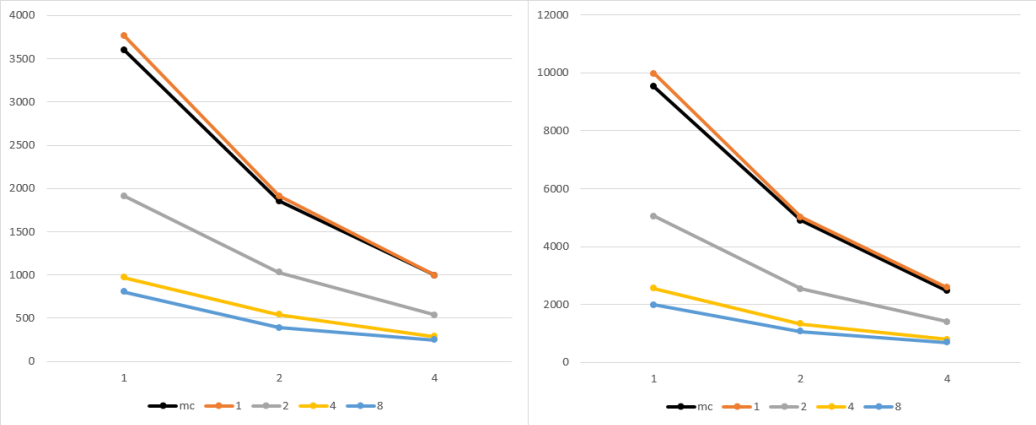


Figure 3.5: Execution time of $parSMURF^1$ and $parSMURF^n$ on the synthetic datasets synth_1 (left) and synth_2 (right). On x axis, the number of threads for each MPI process; on y axis, execution time in seconds; experiments are grouped by the number of MPI processes. Black line is the multi-thread version, while orange, gray, yellow and light blue are the MPI version with 1, 2, 4, and 8 MPI processes.

Both graphs show that the multi-core and the multi-node implementations of *parSMURF* introduce a substantial speed-up with respect to the sequential version (the point in the black line with 1 thread in the abscissa). Note that in Figure 3.5 the black line represents $parSMURF^1$, while the orange line $parSMURF^n$: their execution time is very similar since both use the same overall number of threads, with a small overhead for the MPI version due to the time needed to setup the MPI environment. Table 3.3 shows that the speed-up achieved by $parSMURF^n$ is quasi-linear with respect to the overall number of "aggregated threads" (i.e. the product $n.thr \times n.proc$) involved in the computation, at least up to 16 threads. By enlarging the number of "aggregated threads" we have a larger speed-up, but a lower efficiency, due to the lower number of parts of the partition assigned to each thread, and to the larger time consumed by the MPI data intercommunication.

Table 3.3: Execution time and speed-up of *parSMURF*^{n} with synth_1 and synth_2 datasets. Threads are counted as "aggregated" in the sense that they are the product of the number of MPI processes with the number of threads for each process. All executions have been performed with a 10-fold cross-validation setting.

Aggregated threads	**synth_1 time (s)**	**synth_1 speed-up**
1	3768.59	-
2	1910.19	1.97
4	571.56	3.88
8	542.35	6.95
16	288.82	13.05
32	248.84	15.18

Aggregated threads	**synth_2 time (s)**	**synth_2 speed-up**
1	9981.82	-
2	5020.18	1.99
4	2539.10	3.93
8	1329.74	7.51
16	788.31	12.66
32	686.41	14.54

Results with the synthetic dataset synth_3, that includes 50 million examples, confirm that *parSMURF* scales nicely also when the size of the data is significantly enlarged. Indeed Figure 3.6 (left) shows that by increasing the number of processes and threads we can obtain a significant reduction of the execution time. These results are confirmed by grouping the execution time with respect to the "aggregated" number of threads, i.e. the product $n.thr \times n.proc$ (Figure 3.6 (right)).

Figure 3.7 shows the speed-up (left) and efficiency (right) obtained with this dataset; results are again grouped by the "aggregated" number of threads. Note that with this large dataset we can obtain a larger speed-up, even if, as expected, at the expenses of the overall efficiency.

Different research works showed contradictory results about the comparison of the performance of pure multiprocess MPI, pure multithread OpenMP or hybrid MPI-OpenMP implementations of the same algorithm, showing that several factors, such as algorithms, data structures, data size, hardware resources, MPI and OpenMP library implementations, influence their performance [184, 262, 263, 264, 265, 266].

Regarding our experiments, from Figure 3.6 and 3.8, we can notice how,

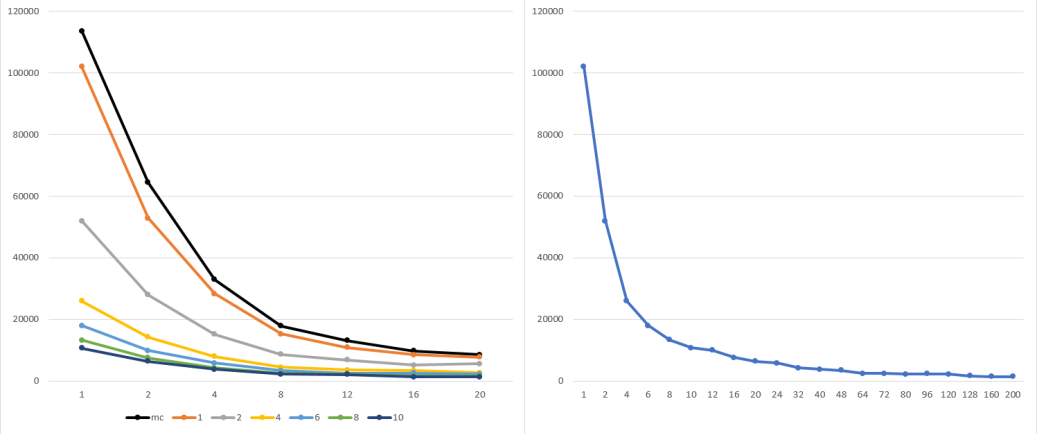


Figure 3.6: Execution time of $parSMURF^1$ and $parSMURF^n$ on the synthetic dataset synth_3. Left: on x axis, the number of threads for each MPI process; on y axis, execution time in seconds. Experiments are grouped by the number of MPI processes. Black line is the multi-thread version, while orange, gray, yellow, light blue, green and blue are the MPI version with 1, 2, 4, 6, 8 and 10 MPI processes. Right: results are grouped by total number of threads ($n.thr \times n.proc$). When a combination is obtainable in more than one way only the best time is considered.

in some cases, a pure MPI implementation may outperform an heterogeneous MPI-multithread or a pure OpenMP-multithread implementation. However, more in general, $parSMURF^n$ allows a higher degree of parallelism, thus resulting in a larger speed-up and efficiency with respect to the pure multi-thread $parSMURF^1$ (Figure 3.7 and 3.9).

Speed-up and efficiency analysis with genomic data

To show how *parSMURF* performs in term of speed-up and efficiency on a real genomic dataset, we carried out the same batch of experiments as in the previous section using this time the Mendelian dataset.

Figure 3.8 shows the execution time of $parSMURF^1$ and $parSMURF^n$ as a function of the "aggregated number of threads", i.e. the product of the number of MPI processes and the number of threads per process. As expected, results show a substantial decrement in execution time with respect to the number of the aggregated threads.

Figure 3.9 shows the speed-up and efficiency of *parSMURF*: on x axis of both graphs, threads are counted as "aggregated", that is the total number of threads is computed by multiplying the number of processes by the number of threads assigned to each process. For the evaluation of speed-up and

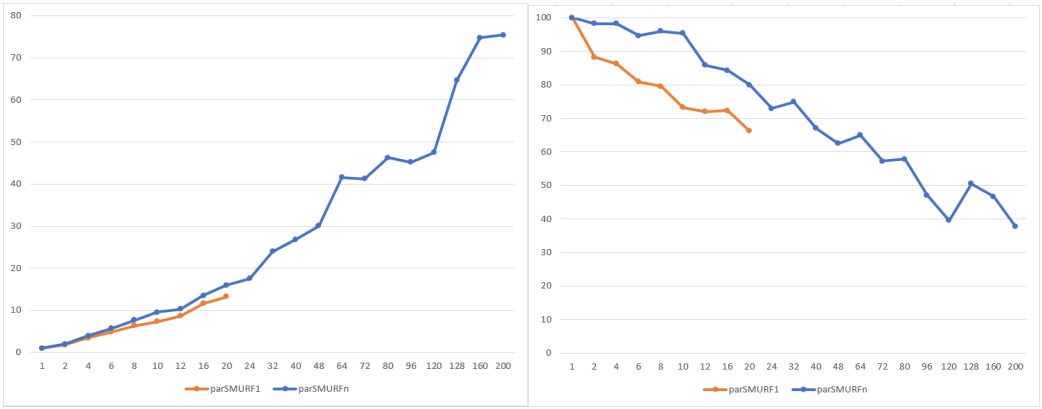


Figure 3.7: Left: Speed-up of *parSMURF*^{1} and *parSMURF*^{n} on the synthetic dataset synth_3. On x axis, the "aggregated" number of threads; on y axis, speed-up. Right: efficiency of *parSMURF* with the synthetic dataset synth_3. On x axis, the "aggregated" number of threads; on y axis, efficiency in percentage. Blue line refers to *parSMURF*^{n}, orange line to *parSMURF*^{1}.

efficiency, *parSMURF*^{1} with only one computing thread has been used as reference for obtaining the computation time of the sequential version.

The maximum speed-up of *parSMURF*^{1} is about 17×, with the execution time decreasing from 97287 seconds of the sequential version to 5695 seconds of the multi-threaded version using 24 cores. The speed-up of *parSMURF*^{n} is even better, with a maximum speed-up of 80× (1181 seconds execution time) obtained using 10 MPI processes with 20 computing threads each. The graph shows that both *parSMURF*^{1} and *parSMURF*^{n} follow the same trend in the increment of the speed-up, but the multi-thread version is limited to 24 threads (each one assigned to a different core), while *parSMURF*^{n} continues this trend up to 288 threads, reaching a speed-up saturation level of 80×. As just observed with synthetic data (Figure 3.7), the efficiency tends to decrease with the number of aggregated threads.

Summarizing both experiments with synthetic and genomic data show that *parSMURF* scales nicely with large data and achieves a significant speed-up that allows its application to big data analysis and to the fine tuning of learning parameters.

Auto-tuning of learning parameters improves prediction of pathogenic non-coding variants

The speed-up introduced by *parSMURF* allows the automatic fine tuning of its learning parameters to improve predictions on real genomic data.

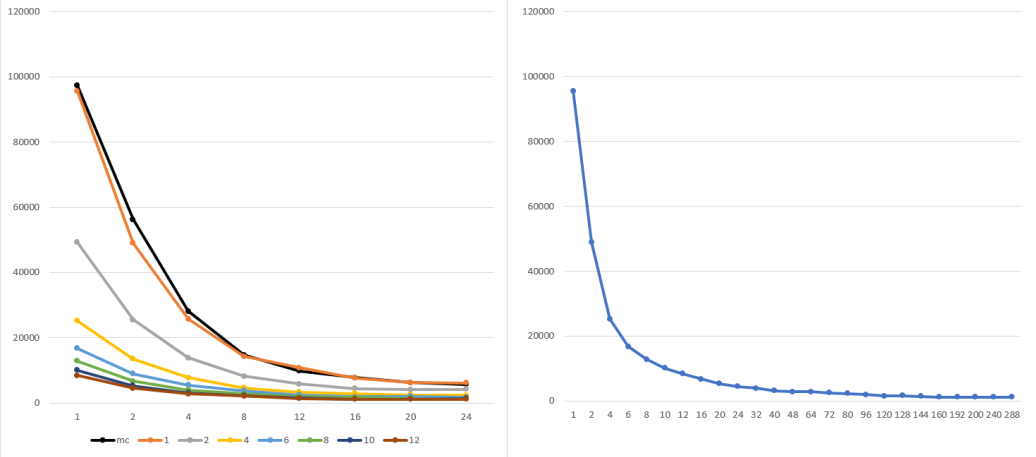


Figure 3.8: Execution time of *parSMURF*^{1} and *parSMURF*^{n} on the Mendelian data set. Left: on x axis, the number of threads for each MPI process; on y axis, execution time in seconds. Experiments are grouped by number of MPI processes. Black line is the multi-thread version, while orange, gray, yellow, light blue, green, blue and brown are the MPI version with 1, 2, 4, 6, 8, 10 and 12 MPI processes. Right: results are grouped by total number of threads ($n.thr \times n.proc$).

Indeed, as preliminarily shown in [254], fine tuning of *hyperSMURF* learning parameters can boost the performance on real data.

To this end we run *parSMURF*^{n} on the Cineca Marconi cluster using auto-tuning strategies to find the best learning parameters for both the prediction of pathogenic non-coding SNVs in Mendelian diseases and for the prediction of GWAS hits that overlap with a known regulatory element.

We compared the auto-tuned results only with those obtained with the default learning parameters of *hyperSMURF*, since our previous studies showed that *hyperSMURF* outperformed other methods, such as CADD [98], GWAVA [101], Eigen [104] and DeepSea [105] with both Mendelian diseases and GWAS hits data [109], and, above all, since we are more interested in showing a proof-of-concept of the fact that auto-tuning of learning parameters may lead to better performances in a real genomic problem.

Generalization performances have been evaluated through an external 10-fold "cytogenetic band-aware" cross-validation (CV) setting. This CV technique assures that variants occurring nearby in a chromosome (i.e. in the same cytogenetic band) do not occur jointly in the training and test sets and thereby biasing results, since nearby variants may share similar genomic features [109]. Learning parameters were selected through a grid search realized through a 9-fold internal CV, that is for each of the 10 training

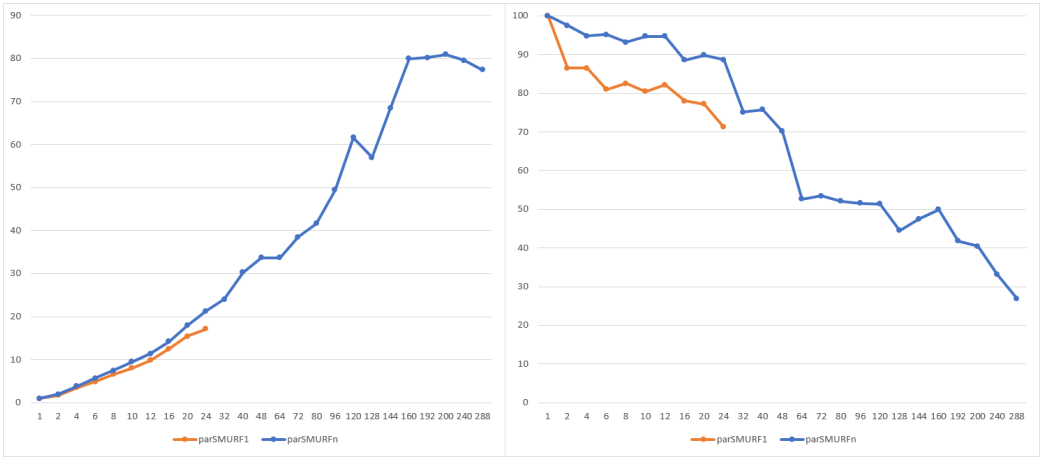


Figure 3.9: Left: Speed-up of of *parSMURF*^{1} and *parSMURF*^{n} with the Mendelian dataset. On x axis, the "aggregated" number of threads; on y axis, speed-up. Right: efficiency of *parSMURF* with the Mendelian dataset. On x axis, the "aggregated" number of threads; on y axis, efficiency in percentage. Blue line refers to *parSMURF*^{n}, orange line to *parSMURF*^{1}.

sets of the external cross-validation, their 9 'cytogenetic band-aware" folds have been used to select the best learning parameters and to avoid putting contiguous variants located in the same cytoband both in training and in the validation set.

This experimental set-up is computationally demanding, but by exploiting the different levels of parallelism available for *parSMURF*^{n} we can obtain a sufficient speed-up to experiment with different hyper-ensembles having different sets of learning parameters.

Performances of the prediction are evaluated via the Area Under the Receiver Operator Characteristic Curve (AUROC) and the Area Under the Precision-Recall Curve (AUPRC). Since data are highly unbalanced, we outline that it is well-known that in this context AUPRC provides more informative results [267, 164].

Improving predictions of pathogenic Mendelian variants

We at first executed *hyperSMURF* with default parameters (specifically: $nParts = 100$, $fp = 2$, $ratio = 3$, $k = 5$, $nTrees = 10$ and $mTry = 5$) in a context of 10-fold cytogentic-band aware CV, as this experiment is used as reference for the next steps.

We tested the auto-tuning feature by performing a grid search over the hyper-parameter space $\mathcal{H}_g$ defined in Table 3.4, central column. Such hyper-

	$\mathcal{H}_g$	$\mathcal{H}_b$
nParts	$\{10, 50, 100, 300\}$	$[10, 300]$
fp	$\{1, 2, 5, 10\}$	$[1, 10]$
ratio	$\{1, 2, 5, 10\}$	$[1, 10]$
k	$\{5\}$	$\{5\}$
nTrees	$\{10, 20, 100\}$	$[10, 100]$
mtry	$\{2, 5, 10\}$	$[2, 10]$

Table 3.4: Hyper-parameter spaces for grid search ($\mathcal{H}_g$) and Bayesian optimizer ($\mathcal{H}_b$) used for the auto-tuning on the Mendelian dataset.

space provides 576 possible hyper-parameter combinations $h \in \mathcal{H}_g$. Then, we applied the auto-tuning strategy based on the Bayesian optimizer, by defining the hyper-parameter space $\mathcal{H}_b$ as in Table 3.4, right column.

To fully exploit the scalability of *parSMURF*, we launched the grid search with the following configuration: 10 instances of *parSMURF*^{n}, one for each fold of the external CV, each one having 10 worker processes, with 6 dedicated threads for processing the different parts of the partition plus further 4 threads for each random forest training and testing. Hence, for the grid-search, we used a total of 2400 CPU cores. Since the Bayesian auto-tuning procedure is less computationally intensive, we chose a more conservative approach on resource utilization for this experimental set-up: we launched one instance of *parSMURF*^{n} having 24 worker processes with 16 threads for the partitions and one for the random forest training and testing. Folds of the external CV are processed sequentially. Therefore, for the Bayesian optimization set-up we used 384 CPU cores.

At the end of this phase, for each optimization strategy, *parSMURF* returns the best hyper-parameter combination for each fold. We then executed 10 repetitions of the external CV using the default parameters, 10 using the best hyper-parameters found by the grid search and 10 using the best hyper-parameters found by the Bayesian optimizer. Performances in terms of AUROC and AUPRC were measured for each repetition and then averaged.

Performance improvements relative to the above parameter tuning experiments and their execution times are summarized in Table 3.5. Results in columns 4 and 5 show a significant improvement in the prediction performances in terms of AUPRC using both optimization strategies (Wilcoxon rank sum test, $\alpha = 10^{-6}$). On the other hand, AUROC is very high in all the experiments, confirming that this metric is not sufficient for the evaluation of prediction performances in the context of highly unbalanced datasets. In Appendix 6, Figures 1 and 2 show the computer ROC and precision-recall curves

of both *hyperSMURF* and *parSMURF*. Also, the Bayesian optimizer proves to be effective in both improving the prediction performances and reducing the computational time: although slightly lower, predictions are comparable to the grid search, but they are obtained at a fraction of the computational power required by the latter. As a matter of fact, the CPU time required by the entire grid search counted more than $120k$ hours, compared to $16k$ hours used by the Bayesian optimization strategy.

Table 3.5 reports th average AUROC and AUPRC measured on the training sets: results show that the ratio between training and test AUROC or AUPRC is quite similar between *hyperSMURF* and *parSMURF*, and even if, as expected, results on the training sets are better, they are comparable with those obtained on the test data. These results show that performance improvement is not due to overfitting, but to a proper choice of the hyper-parameters well-fitted to the characteristics of the problem.

Table 3.5: Summary of performance improvements obtained by *parSMURF* by tuning the learning parameters on the Mendelian dataset. Results are averaged across 10 repetitions of the 10-fold cytoband-aware cross-validation. "AUROC avrg" and "AUPRC avrg" are averaged across the 10 folds; standard deviation in brackets. Columns 2 and 3 report AUROC and AUPRC metrics on the training set, columns 4 and 5 report the same metrics evaluated on the test set. Default parameters: nParts 100, fp 2, ratio 3, nTrees 10, mTry 5.

	training AUROC mean (std)	**test AUROC mean (std)**
Default parameters	0.99958 (0.00005)	0.99281 (0.00032)
Grid search	0.99986 (0.00009)	0.98968 (0.00140)
Bayesian optimizer	0.99989 (0.00011)	0.99264 (0.00043)

	training AUPRC mean (std)	**test AUPRC mean (std)**
Default parameters	0.53143 (0.02714)	0.42332 (0.00391)
Grid search	0.60023 (0.15977)	0.47025 (0.00585)
Bayesian optimizer	0.65388 (0.22123)	0.46153 (0.00302)

To assess whether the difference in performance between *hyperSMURF* and *parSMURF* can be related to their different capacity of selecting the most informative features, we measured Spearman correlation between both *hyperSMURF* and *parSMURF* scores with each of the 26 features used to train the hyper-ensembles for all the examples of the dataset. Table 3 in Appendix

6 reports the correlation between the true labels of the examples and the predictions obtained by *hyperSMURF* using the default set of hyper-parameters, *parSMURF* with the default, grid optimized and Bayesian optimized set of hyper-parameters. We found that *hyperSMURF* and *parSMURF* achieved very similar Spearman correlation on each feature (the Pearson correlation between the vectors of Spearman correlations of *hyperSMURF* and *parSMURF* is about 0.98). Both *hyperSMURF* and *parSMURF* obtained the largest Spearman correlation coefficients for features related to the evolutionary conservation of the site (e.g. vertebrate, mammalian and primate PhyloP scores) and for some epigenomic features (histone acetylation, methylation and DNAse hypersensitivity). Again, these results show that it is unlikely that the improved performance of *parSMURF* can be explained through its better capacity of selecting the most informative features, but instead by its capacity of auto-tuning its learning hyper-parameters and its capacity to find a model that better fits the data.

In addition, in Table 3.6 some examples of pathogenic variants which have been ranked remarkably better by *parSMURF* than *hyperSMURF* are reported. Further details about these variants are shown in Table 6 of Appendix 1.

Table 3.6: Examples of pathogenic Mendelian variants better ranked by *parSMURF* with respect to *hyperSMURF*. The first two columns report the chromosomal coordinates, while the last two the difference in ranking between respectively *parSMURF* with grid search ($\mathcal{H}_g$) and with Bayesian optimizer ($\mathcal{H}_b$) with respect to *hyperSMURF*. The larger the absolute difference, the higher the improvement (see also Table 6 in Appendix 6 for more information).

chr	**pos**	hS rank $-\mathcal{H}_g$ rank	hS rank $-\mathcal{H}_b$ rank
1	100661453	2308597	169786
3	12421189	663054	421027
X	138612889	194290	111499
13	100638902	70175	69069
6	118869423	63078	55789
16	31202818	50539	103623
12	121416444	21848	65773

Prediction performances of *parSMURF* with an independent Mendelian test set

We collected novel Mendelian pathogenic variants by adding 64 newly positive (pathogenic) non-coding variants manually annotated according to a comprehensive literature review. We included only those variations and publications judged to provide plausible evidence of pathogenicity (Appendix 6, Table **??**). As negatives we used common variants downloaded from NCBI [268], i.e. variants of germline origin and having a minor allele frequency (MAF) ≥ 0.01 in at least one major population, with at least two unrelated individuals having the minor allele, where major populations are those included in the 1K genome project [17]. We selected only those variants that lie in non-coding regions using Jannovar [269]. The final number of negatives (about 3 millions of examples) has been randomly sampled in such a way that the ratio positives/negative in the original and in the new Mendelian data set used for validation is approximately the same. Both the positive and negative examples have been annotated with the same 26 genomic and epi-genomic features used for the original Mendelian data set. We trained *hyperSMURF* and *parSMURF* on the overall original Mendelian data set and then we tested the resulting models on the unseen separated new Mendelian data set used for validation. Since the new positive set also contains small insertions and deletions, similarly to [108], to predict the pathogenicity of the deletions, we used the maximum score of any nucleotide included in the deleted sequence, while for insertions we used the maximum score computed for the two nucleotides that surround the insertion. Results with the independent Mendelian test set show that the models are able to obtain relatively high AUPRC results, especially when *parSMURF* is applied, showing that our models can nicely generalize. Also with this new independent data set *parSMURF* significantly outperforms *hyperSMURF* (Table 3.7).

Model	AUROC	AUPRC
hyperSMURF	0.945216	0.098153
parSMURF- Grid Search	0.941026	0.409067
parSMURF- Bayesian optimizaion	0.928158	0.192568

Table 3.7: *hyperSMURF* and *parSMURF* (with Grid Search and Bayesian optimization) prediction performances obtained over a fully independent Mendelian test set composed by newly annotated pathogenic variants (positive examples) and common neutral variants (negative examples).

Improving predictions of GWAS hits

A similar experimental setup has been employed for improving the predictions of GWAS hits. At first we executed *parSMURF* with the default parameters as reference for the next batches of experiments. Then, we tested the auto-tuning feature by performing a grid search over the hyper-parameter space $\mathcal{H}_g$ defined in Table 3.8, central column. Such hyperspace provides 256 possible hyper-parameter combinations $h \in \mathcal{H}_g$. Next, we tested the Bayesian optimizer by defining the hyper-parameter space $\mathcal{H}_b$ as in Table 3.8, right column.

Table 3.8: Hyper-parameter space for Grid search and Bayesian Optimization used for auto-tuning *parSMURF* on the GWAS data set.

	Grid-search	Bayesian-opt.
`nParts`	$\{10, 20, 30, 40\}$	$[10, 40]$
`fp`	$\{1, 2, 5, 10\}$	$[1, 10]$
`ratio`	$\{1, 2, 5, 10\}$	$[1, 10]$
`k`	$\{5\}$	$\{5\}$
`nTrees`	$\{10, 20, 50, 100\}$	$[10, 100]$
`mtry`	$\{30\}$	$\{30\}$

Results are shown in Table 3.9. As for the Mendelian dataset, AUROC is very high in all experiments. On the other hand, test results show a significant increase of AUPRC with both auto-tuning strategy, with the grid search leading a better outcome than the Bayesian optimizer. Figures 3 and 4 in Appendix 6 show the ROC and precision-recall curves of *hyperSMURF* and *parSMURF*.

These results further show that fine tuning of learning parameters is fundamental to significantly improve prediction performances, showing also that *parSMURF* is a useful tool to automatically find "near-optimal" learning parameters for the prediction task under study.

Assessment of the effect on prediction performance of the variants imbalance across regulatory regions.

As recently pointed out in [110], pathogenic scores predicted by several state-of-the-art methods are biased towards some specific regulatory region types. Indeed also with Mendelian and GWAS data the positive set of variants is located in different functional non-coding regions (like 5'UTR, 3'UTR or Promoter) and is not evenly distributed over them. This is also the case for the negative set (see Tables 4 and 5 in Appendix 6). Because of this

imbalance, performances of different categories are different as already mentioned by Smedley et al. for the ReMM score on the Mendelian data [108]. It is possible that our *parSMURF* parameter optimization will favor different categories because of the number of available positives and the different imbalance between positives and negatives across different genomic regions. To show that the optimization is robust to this characteristic of the data we compared performances on each genomic category before and after parameter optimization. Variant categories have been defined through Jannovar [269] using the RefSeq database.

Then we retrained and re-optimized the parameters on a training set using cytoband-aware cross-validation, where all categories have the same imbalance by subsampling negatives to the smallest imbalance of the categories. More in detail, we used the following strategy: (1) sub-sample the negatives to the same imbalance in all categories. Mark the variant if it is in this new subset; (2) partition the whole dataset into 10 folds as done previously; (3) for each training step select only the previously marked variants of the 9 training folds; (4) sub-sample the test set using the same categorization and ratios as in (1). To take into account the variability between runs, we repeated this process 10 times for the Mendelian dataset and 5 times for the GWAS dataset. Using this strategy both training and test sets are equally "per region balanced", so that category unbalance is kept under control and we can correctly evaluate whether our approach may unnaturally inflate predictions towards a specific region due to the original per-region imbalance of both datasets.

Table 3.9: Summary of the performance improvements obtained by *parSMURF* by tuning its learning parameters with the GWAS Catalog dataset. Results are averaged across 10 repetitions of the 10-fold cytoband-aware cross-validation. "AUROC avrg" and "AUPRC avrg" are averaged across the 10 folds. Default par: nParts 100, fp 2, ratio 3, nTrees 10, mTry 5.

	AUROC average	**AUROC stdev**
Default parameters	0.99426	0.00169
Grid search	0.99459	0.00174
Bayesian optimizer	0.99346	0.00193

	AUPRC average	**AUPRC stdev**
Default parameters	0.48058	0.07138
Grid search	0.72533	0.03616
Bayesian optimizer	0.71945	0.03675

Results are shown in Figures 5 and 6 of Appendix 6: for all variant categories we see a performance gain or similar performance in *parSMURF* with respect to *hyperSMURF* for both the Mendelian and GWAS data set, suggesting that *parSMURF* is robust to the categorical composition of the variants. Moreover in the "per region balanced" setting AUPRC results are systematically better with the Mendelian data set (Figure 5, Appendix 6) and quite always better or comparable with the GWAS data (Figure 6, Appendix 6). These experimental results show that both *hyperSMURF* and *parSMURF* can properly handle different imbalances of variant categories, by using "smart" balancing techniques on the training set able to both balancing and at the same time maintaining a large coverage of the available training data. The increase of performance of *parSMURF* with respect to *hyperSMURF* is not driven by the under- or over-representation of variants belonging to a particular region type, but by its capacity of automatically fine-tuning the set of its hyper-parameters, according to the given task at hand.

Analysis of the hyper-parameters

Since we adopted cross-validation techniques to estimate the generalization performance of the models, we averaged the best parameters values separately estimated for each fold, in order to obtain a single set of optimal parameters. Tables 1 and 2 of Appendix 6 show the sets of best hyper-parameters found by both the optimization techniques with the Mendelian and GWAS datasets.

Of the 6 hyper-parameters, we noticed that *nParts*, *fp* and *ratio* are the main factors that drive the performance improvement. *Fp* and *ratio* hyper-parameters provide the rebalancing of the classes. A larger *fp* value translates into a larger number of positive examples generated through the SMOTE algorithm (see Section Methods), thus reducing the imbalance between positive and negative examples in the training set: Tables 1 and 2 of Appendix 6 show that enlarging the ratio of novel positive examples *parSMURF* improves results over *hyperSMURF*, and confirm that fine-tuned balancing techniques can improve the results. The *ratio* hyper-parameter controls the ratio between negative and positive examples of the training set. Results in Tables 1 and 2 of Appendix 6 show that values larger than the default ones improve performance, since in this way we can both reduce the imbalance between negatives and positives (for the Mendelian data sets we move from $36000:1$ to $10:1$, and for GWAS from $700:1$ to $10:1$), and at the same time we maintain a relatively large coverage of the negative data (in each partition negative examples are sampled in such a way to obtain ten negatives for each positive of the training set).

The results also show that a larger coverage of negative examples is obtained by incrementing the *nParts* hyper-parameter, since by increasing the number of partitions, less negatives are discarded. Moreover more random forests are trained thus improving the generalization capabilities of the hyper-ensemble. Finally, for the GWAS dataset, the *mtry* hyper-parameter plays a fundamental role in the increment of the performance, due to the high number of features of the dataset. Overall, the analysis of the hyper-parameters confirms that their fine tuning is fundamental to improve the performances of the hyper-ensemble.

3.2 parSMURF-NG and ParBigMen

Encouraged by the promising results of *parSMURF*, we further developed its core concepts to increase its scalability and prediciton accuracy. In this section we present a newly developed version of *parSMURF* called *parSMURF-NG*(ParSMURF New Generation), together with its future application in the context of precision medicine: we plan to expand the original Mendelian dataset with new pathogenic variants and genomic features, and incorporate *parSMURF-NG* in the next version of Genomiser [108]. This, however, requires a massive amount of computational power: for this reason, we applied to the 21^{st} PRACE (Partnership for Advanced Computing in Europe) call, and the project was awarded with 50 million computing core-hours for pursuing these goals.

This section summarizes the motivations and technical details of the whole project, called *ParBigMen* (ParSMURF application to Big genomic and epigenomic data for the detection of pathogenic variants in Mendelian diseases), as illustrated in the PRACE project proposal.

3.2.1 Motivation

Starting from the premises introduced by *parSMURF*, we already established that variant identification and analysis of NGS data plays a central role in genomic and personalized medicine, and has been finally made feasible by high-throughput bio-technologies and Artificial Intelligence. We also already stated that while disease-associated variants which fall in protein-coding areas of the DNA are fairly well studied [243, 244], the understanding of the impact of variants occurring in the non-coding regions of the genome is largely incomplete. Nonetheless, several studies showed that most of the potential pathogenic and deleterious variants do not lie in the coding areas of the genome [246], hence the scientific community started to shift its attention

towards the understanding of non-coding portions of the genome.

However, facing this task with traditional Machine Learning methods has been proven to be particularly challenging for several reasons, most of which are related with the sparsity of pathogenic mutations among the "sea" of the neutral variants: from a machine learning standpoint, this ultimately translate to a high unbalance between classes of examples to be learned, leading to a very challenging classification problem.

The imbalancing problem has been addressed in *hyperSMURF* which, compared to other approaches, reaches state-of-the-art results in terms of accuracy of predictions when dealing with extremely unbalanced datasets. *hyperSMURF* represents the evolution or ReMM (Regulatory Mendelian Mutation), the ML core of Genomiser [2], a whole genome analysis framework which is able to score the relevance in terms of pathogenicity and deleteriousness of a variant in the non-coding portions of the genome, and associate regulatory variants to specific Mendelian diseases. In Genomiser, *hyperSMURF* is trained with a dataset composed of 406 manually curated pathogenic Mendelian SNVs and more than 14 million of neutral SNVs, all belonging to non-coding genome. Each SNV is annotated with 26 genomic features, including conservation scores, transcription and regulation annotations, DNAse hypersensitivity and more. In the Genomiser framework, *hyperSMURF* trained with this dataset is called ReMM model (Regulatory Mendelian Mutation) and a score is assigned to each SNV (ReMM score), predicting the deleteriousness of each position. As Genomiser and the ReMM scores have been extensively used by the scientific community working in this field, our proposal represents a natural follow up of the research work performed for these tools, and it is aimed to improve the identification of new pathogenic SNVs in non-coding regions. However, computational requests for advancing this research are very high and they are only feasible with the use of a HPC system.

Originally, ReMM scores were evaluated with the *hyperSMURF* initial software implementation, which was written in the R programming language; due to its inefficiency, we only managed to perform a very brief tuning of its learning hyper-parameters. However, in [254] we showed that *hyperSMURF* hyper-parameter tuning plays a major role in providing accurate predictions. For overcoming the limitations of *hyperSMURF*, we developed *parSMURF* - as discussed in section 3.1 - that thanks to its efficient implementation aimed to multicore and multinode architectures, and several auto-tuning features, is able to improve the quality of the learner, thus improving the predictions altogether. This, however, required a dedicated infrastructure to be accomplished: we used Cineca Marconi SKL computing power for the auto-tuning of the learner hyper-parameters over the Mendelian SNV

dataset, reducing the computational time from 18 years (estimated - using a single core machine) to less than two weeks: for this reason, such task cannot be faced without an adequate computing power that nowadays is available only on HPC facilities.

As a natural evolution of the work performed on ReMM, *hyperSMURF* and *parSMURF*, we developed a newer version of the algorithm, called *parSMURF-NG*, with several goals in mind, more in detail:

- the release of the highly parallel *parSMURF-NG* application able to scale with big data and to fully exploit the High Performance Computing architectures for relevant prediction problems in the context of Personalized and Precision Medicine
- the application of *parSMURF-NG* to big omics data, where a large number of features will be investigated and used to predict pathogenic variants. We expect to obtain breakthrough models able to achieve a significant advance in state-of-the-art prediction of pathogenic variants in Mendelian and complex genetic diseases.
- the evaluation and release of new ReMM scores to the entire scientific community for the prioritization of pathogenic and deleterious SNVs.

We expect to achieve breakthrough results in the prediction of Mendelian pathogenic variants, and to embed *parSMURF-NG* into Genomiser, the state-of-the-art tool for the molecular diagnosis of Mendelian diseases, in collaboration with the US Jackson Lab.

We also want to remark the interdisciplinarity of this approach, as its impact may be relevant in different contexts: starting from the method and its implementation, we point out that its high level of parallelism and its programming model may be taken as a paradigm for introducing a new class of automatic learning methods specifically designed for HPC infrastructures. Also, *parSMURF-NG* is designed for genomic problems, but it can easily be adapted for dealing with different kinds of highly imbalanced dataset and for providing different classes of outcomes, hence expanding its usability in a broader range of contexts. Finally, the impact of the newly predicted scores can transcend its usage in genomic and precision medicine, as nowadays even more medical research studies uses heterogeneous data, explicitly taking into account inside its mathematical model the role of genomic pool.

However, even by using the newly developed and more efficient *parSMURF-NG*, this project requires exceptional computational resources, only available in specialized HPC facilities. To this end, we applied to the 21^{st} PRACE open access call for high impact scientific discovery and engineering research & development: this project is among those that hava been awarded with Tier-0

computing resources on European High Performance Computing centers, and we have been granted 50M core-hours on the SuperMUC-NG partition of the Leibniz Supercomputing Centre in Garching (Munich, Germany). This supercomputer is in the top 10 ranking of the current TOP500 list; also, the equivalent estimated value of the awarded computing time is approximately half a million Euros.

3.2.2 From *parSMURF* to *parSMURF-NG*

parSMURF-NG has been developed from the ground up taking *parSMURF* as reference and, at the best of our knowledge, is the only HPC-oriented machine learning software able to cope with extremely unbalanced genomic dataset. *parSMURF-NG* is the parallel version of *hyperSMURF* and the more efficient and Big Data oriented version of *parSMURF*. The effectiveness of the learning strategy common to all implementations has been shown in [109] where we compared our learning approach with several other tools for automatic learning applied to genomic and personalized medicine: we showed that *hyperSMURF* outperformed all the considered alternative tools. Also, *hyperSMURF* has been used to evaluate Regulatory Mendelian Mutation (ReMM) scores for relevance prediction of non-coding variations in the human genome [108] used in the state-of-the-art tool Genomiser. For these reasons, we believe that *parSMURF-NG* will be the ideal tool for identifying pathogenic and deleterious SNVs, given the expanded dataset with new genomic features, and to evaluate new ReMM scores.

From a technical standpoint, *parSMURF-NG* is written in C++. The choice of this programming language is given by the pursuit of uncompromised performance, efficiency and optimal memory usage, as the algorithm is very CPU intensive and the size of dataset to be processed is in the order of tens of Gigabytes; such goals are not really obtainable with other programming languages: as a matter of fact, the original *hyperSMURF* is implemented in the R programming language and the difference in efficiency and execution times is tenfold. Also, using C++ allowed us to fine tune the performance of the implementation depending on the underlying hardware configuration: the code is optimized for standard x86-64 processors, but, for instance, it also exploits the additional features of Intel XeonPhi CPUs. Also, interfacing with the MPI library and manually managing each MPI operation allowed us to fine tune synchronization and intercommunication between processes.

A small part of the code – the Bayesian Optimizer core – is written in Python. This is justified by the fact that one of the strategies for finding the optimal set of learning hyper-parameters is based on Bayesian Optimization.

One popular, effective and highly configurable package for this task is *skopt*, included in the SciKit Learn library, also written in Python. We also considered the package *spearmint*, as in the previous version of *parSMURF*, but its difficulty in configuration and usage made us prefer skopt.

parSMURF-NG is mostly self-contained and it does not require particular libraries or external applications. This makes *parSMURF-NG* code extremely portable and can be compiled on any platform that provides a C++11/14 compiler and an implementation of the MPI library (mandatory). The main machine learning strategy of *parSMURF-NG* is the random forest algorithm. We use the Ranger library[1] to provide this functionality. Also, K-Nearest Neighbor search strategy for the SMOTE oversampling method is provided by the ANN library[2]. Code of both libraries is included in the *parSMURF-NG* repository, as they have been heavily modified for the use in our codebase. *parSMURF-NG* relies on a few utility libraries, namely Easylogging++[3], Zlib[4], jsoncons[5] and getopt. Code for these libraries is downloaded at compile time. Bayesian Optimization is done through the skopt Python package. This requires Python 3.5 (or newer), but it is not mandatory if Bayesian Optimization is not performed.

One of the major constraint that limited the scalability of *parSMURF* over hundreds of computing nodes was its master-slave(s) programming paradigm: under this model, a single master process orchestrates a pool of workers. Although being a rational model, as the master process manages the data to be sent to and received from each worker and also manages the execution flow orchestrating the execution of the workers, this imposes a severe limit to the scalability, as the workload on the master process increases with the number of workers. As an extreme case, several workers may be idle, waiting the master process to send them the data to be processed, thus affecting the global efficiency of the approach.

On the opposite, *parSMURF-NG* has been re-designed to circumvent this limitation: we chose a parallel programming model where a set of processes independently work on several subsets of the dataset. In this model, a relevant coordination and synchronization occurs only when the final results need to be collected, hence maximizing efficiency and scalability. This approach guarantees that each process is a worker and no orchestration by a master is required. In almost all the execution flow, each worker is in charge of autonomously read the data that it needs for performing the assigned com-

[1] `https://github.com/imbs-hl/ranger`
[2] `https://www.cs.umd.edu/~mount/ANN/`
[3] `https://github.com/amrayn/easyloggingpp`
[4] `https://zlib.net/`
[5] `https://github.com/danielaparker/jsoncons`

putation, and is in charge as well of synchronizing itself with the rest of the pool: this has the dramatic effect that, by increasing the number of workers, scalability is not affected (if not marginally), hence hundreds of processes - and hundreds of computing nodes - can efficiently work for solving a single task. However, we have to remark that, although being minimal, additional synchronization is required during parameters tuning.

In *parSMURF-NG* different strategies have been employed to reach a high degree of parallelization at nested levels: computation is distributed across nodes of an HPC facility using the MPI library, which also manages intercommunication between processes. Inside each node, parallelization is performed through threading by using the low-level C++ 11/14 STL threads library. The previous version of *parSMURF* relied on OpenMP 2, but experimental results showed that this approach did not scale well with the number of computing threads, due to major overheads in scheduling and threads management. On the contrary, C++ threads library has been proven to be very efficient even in architectures featuring a very high core-per-package count, such as Intel XeonPhi CPUs. Lowest level of parallelization occurs at instruction level, as the access pattern for data from memory is optimized so that x86-64 vector instructions can be used. Also, code is optimized so that automatic vectorization and unrolling of loops is performed at compile time: this is particularly important in Intel XeonPhi processors, as by exploiting 512-bit registers and AVX-512 instruction set allows major speed-ups.

I/O usage during the execution of *parSMURF-NG* is time-wise limited, as access to storage and networking occurs only at the beginning of the execution, when each process reads the data from the (network) file system, and at the end of the computation, when results are gathered to rank 0 and saved to disk; this last operation, however, does not represent a bottleneck in computation as, due to the small size of the output file, requires a few seconds to be completed (tests performed on real HPC systems). Additional I/O access due to synchronization occurs in hyper-parameter tuning. However, data import can stress the I/O infrastructure a lot, as the entire pool of MPI processes accesses the dataset at the same time and must import a great amount of data. This impacts the performance of the I/O stage especially in Network File Systems (NFS) as those used in all major HPC facilities. For this reason, I/O operations (disk read and write, network access) involve the use of the MPI library and are optimized to use the underlying hw infrastructure. Preliminary tests with non-MPI disk operation functions (standard C++ STL) which disregards the HPC optimized I/O capabilities turned into a 75× increase of import time in the Marconi KNL system (20 seconds vs. 25 minutes for importing 12GB of data over 128 MPI processes). Hence, we cannot stress enough how an optimized I/O infrastructure is relevant in keep-

ing the execution time low. The dataset is composed of two files: a matrix in plain text or binary format, and a labeling file. Depending on the problem, they can span from a few KiloBytes to dozens of GigaBytes. Results are stored in a single file containing a vector of floating-point numbers (one for each sample in the dataset).

One of the tasks of the project - actually, the first - is dataset preparation as the rare Mendelian Single Nucleotide Variant dataset needs to be expanded with new features. This requires raw data downloads from publicly available repositories and a small pre-processing - i.e. aggregation of several datasets and some elementary filtering and statistical (average, minimum, maximum, etc...). This dataset enrichment, data upload and feature selection will be performed in batches: once a sufficient number of features have been added to the SNV dataset, this is transferred to the data center for feature selection with the wrapper method (*parSMURF-NG* itself).

3.2.3 ParBigMen: a project to boost prediction of pathogenic Mendelian variants

In this PRACE project, we will use the newly implemented *parSMURF-NG*, which further improves the efficiency and scalability of our approach, including at the same time all the functionalities of *parSMURF*. As *parSMURF-NG* is already in production stage and has already been extensively tested on several HPC facilities, the pursuit of the goals of this project requires two macro tasks:

1. the creation of an enlarged new dataset of Mendelian SNV
2. the production of new ReMM scores by finely tuning *parSMURF-NG* on the Mendelian dataset.

For task 1., the starting point is the original Mendelian SNV dataset initially used for evaluating the ReMM scores. This dataset is composed of more than 14 million SNVs, out of which only 406 are marked with very high probability to be pathogenic or deleterious, with evidence from state-of-the-art literature. Each SNV is characterized by 26 genomic attributes comprehensive of conservation scores, DNAse hypersensitivity, transcription and regulation annotations, methylation and more. From a computational standpoint, the original dataset is a matrix of 14M x 26 floating point elements.

Expanding this dataset requires the identification of new deleterious or pathogenic SNVs and the annotation of each SNV with new genomic attributes (features). Adding new deleterious SNVs not only reduces the im-

balance between classes, but also introduces novel essential information about pathogenic variants located in different regulatory regions, thus enlarging the availability of positive SNVs to train the ensemble model. Moreover, adding new genomic attributes to the Mendelian dataset overcomes one of the limitations of the original approach, that is training a model with a dataset of few genomic features for reducing the empirical time and space complexity of the task.

More in detail, regarding the addition of newly discovered pathogenic or deleterious SNVs, we already identified more than 80 new variants that are going to be added to the dataset, thanks to the collaboration with the Jackson Lab. As for the set of the new genomic features, we plan to use omics features from the ENCODE Project[6], UCSC Genome[7] and the International Human Epigenomic Consortium[8]. We will enlarge by two orders of magnitude the number of informative genomic attributes to better characterize the deleterious and pathogenic SNVs in order to further improve the quality of the predictions.

Specifically, preparing the raw features requires to download the raw data from the available repositories, and pre-process and aggregate them, as data in these repositories are subject and tissue specific. Numerical data preprocessing can be performed by applying simple statistics, such as mean, standard deviation, minimum, maximum, etc. After pre-process, the resulting feature is a vector of floating point values that will be added to the existing SNV dataset.

Finally, as not all the computed features are equally informative, some methods of feature analysis and selection must be applied, either by running the *parSMURF-NG* application with the new set of features and compare to the existing results, or independently from the application, by applying uni and multivariate techniques for feature relevance analysis.

The whole process can be executed iteratively, as we can collect data, generate a set of features (for instance 50), test them using a wrapper method (i.e. with *parSMURF-NG* itself), keep only the most informative and then repeat the process with the newly enriched dataset. We remark that the most time consuming operation in these steps is the features evaluation with *parSMURF-NG*, as the download and pre-process steps are almost sequential and can be automated with the use of Python scripts. Incidentally, evaluation with *parSMURF-NG* is also the step that mostly requires the usage of Tier-0 resources, while the rest can either be executed on the HPC facility - this

[6] `https://www.encodeproject.org/`

[7] `https://www.genome.ucsc.edu/`

[8] `https://ihec-epigenomes.org/`

may come handy for preserving the raw data in case further analysis might be required - or offline. We also remark that as the first batch of features is ready, we can start using *parSMURF-NG* at its full potential: this means that HPC usage will start within the first two weeks from the project allocation.

The task 2. will be performed when a relevant and highly informative set of features is added to the Mendelian SVN dataset. In this step we will use this newly constructed dataset with *parSMURF-NG* for producing a new set of ReMM scores. This is the most computing intensive task of the whole project, as we need to fully explore the hyper-parameter space with the auto-tuning technique for ensuring the best possible performance of the learning algorithm. For this reason, the use of Tier-0 resources for this task is mandatory, to provide the computational infrastructure needed for running such a complex task in a feasible amount of time. We remark that task 2. can also start before the end of the feature selection stage, as we can also consider the SNV dataset composed by all the new evaluated features - i.e. without feature selection. In this way, at the end of task 2. we will have two different models: one generated by training *parSMURF-NG* with the dataset comprising all the available features, and one by training *parSMURF-NG* with the dataset comprising a subset of asserted most informative features.

3.2.4 Scalability results with *parSMURF-NG*

At present time, we extensively tested the scalability performance of *parSMURF-NG* in two datacenters featuring different hardware architectures. As this is a proposal for a future work, we do not have results concerning the evaluation of the new ReMM scores. However, performance of the learning strategy has been validated by repeating the tests over the Mendelian and GWAS datasets presented in the previous section, obtaining the same results, thus confirming the state-of-the-art preformance of the algorithm.

The experiments presented in this section have been performed in the context of PRACE Access calls: in special calls, research groups are awarded with a limited amount of computing core-hours on HPC infrastructure of choice, with the aim of testing the scalability of software solution that will concur in the actual PRACE calls. For us, we applied to the PRACE Type A Preparatory Calls 2010PA5046 and 2010PA5332, both awarded with 100000 core-hours on Cineca Marconi KNL partition for the first call) and 50000 on the HLRS Apollo HAWK for the second.

Results have been officially presented on the two final reports that have been submitted to the PRACE consortium. In this section we report the main scalability results obtained with synthetic data.

Scalability of *parSMURF-NG* over synthetic data

The production code of *parSMURF-NG* has been extensively tested on multiple HPC infrastructures. As a matter of fact, we evaluated its scalability performance under a synthetic dataset on Cineca Marconi KNL and HLRS Apollo Hawk partitions. Code has also been tested and used on Cineca Galileo and Marconi SKL partitions, but no official scalability tests have been performed there. Hardware specifications of the two systems are summarized in Table 3.10.

Table 3.10: Summary of the specification of the HPC systems used for evaulating the scalability of *parSMURF-NG*.

	Cineca Marconi A2 KNL partition	**HLRS Apollo HAWK partition**
CPU Family	Intel Xeon-Phi Knights Landing	AMD Rome
CPU Specs	68 cores (4x hyper-threading) 1.4 GHz	64 cores (2x hyper-threading) 2.24 GHz
CPUs per node	1	2
Memory per node	96 GBytes	256 GBytes
Number of nodes	3600	5632
Interconnection: brand, topology and speed	Intel Omnipath 2:1 Fat-tree 100GBit/s	Intel InfiniBand HDR200 Enhanced 9D-Hypercube 200 GBit/s

On both systems, we followed the gold standard rules for assessing the performance, that is evaluating both strong and weak scalability. The former requires to solve for several times a problem of fixed size, each time by gradually increasing to computational power assigned to the task. This means, creating a single dataset and running different simulation with an increasing numbers of CPU cores or computing nodes. Ideally, the speed-up would linearly increase with the number of added computing resources, i.e. if n cores require time t for solving a problem, using $2n$ cores would require $t/2$ time, hence obtaining a speed-up of 2x.

On the other hand, weak scalability is a different approach for measuring the performance of a parallel implementation, as this requires to solve several times a problem whose size is proportional to the employed computational power. This means creating different datasets of increasing size and solving the respective problem employing a likewise increasing amount of computational resources. Ideally, the computing time among runs would be the same,

i.e. if a problem of size d is solved in t time using n CPU cores, a problem of size $2d$ would be solved again in t time by using $2n$ CPU cores.

To insure comparability of the results, we performed the same tests on both systems, as the task to be solved was the same in terms of algorithm parameters and hyper-parameters. The only difference was in the number of computing threads assigned to each MPI process, as AMD Rome architecture manages less threads per package than Marconi's Intel XeonPhi processors. This also explains the difference in total computing time. The software was compiled with Intel ICC on Marconi KNL and with AMD AOCC on HLRS Hawk, using the same set of optimizations (whenever possible).

Scaling tests were executed by launching the same configuration several times, each time increasing the number of computing nodes from 1 to 128 following the power-of-2 law. On Marconi KNL, each MPI rank was instructed to launch 8 threads for partition processing, each of them managing 24 threads for random forest train and test and 4 threads for oversampling of the minority class, hence the total number of threads on each node is 8*(24+4)=224. To stress-test the inter-nodes intercommunication capability offered by the infrastructure, we forced the allocation of MPI processes so that each MPI rank was assigned to a different node.

On Hawk, each MPI rank was instructed to launch 8 threads for partition processing, each of them managing 16 threads for random forest train and test and 1 thread for oversampling of the minority class, hence the total number of threads on each node is 8*16=128. As each Hawk node features two 128-cores processors, two MPI processes have been assigned to each node.

For the strong scalability test we used a matrix of 30M rows and 100 columns, with a label file of 30M elements. Out of the 30M samples, 2000 are marked as belonging to the minority class (1) and the rest are marked as negatives (0) in the label file. Out of the 100 features, only 20 are informative of each sample class, while the rest is randomly generated.

For the weak scalability test we generated three groups of datasets, each one with a different number of rows and different proportion of positives. However, all datasets have the same number of columns (50), out of which only 15 are significant. Datasets for weak scaling are summarized in Figure 3.10. All the datasets used in the following tests are similar in size to the final dataset that will be used in the proposed project.

Strong scaling

As before, for the evaluation of strong and weak scaling we followed the classic experimental setup for the assessment of the performances of parallel algorithms [261].

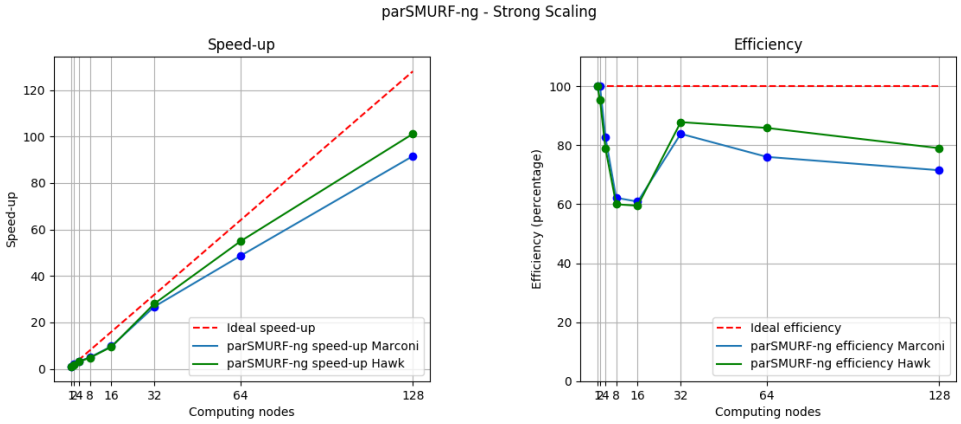
N of MPI processes	Minority class samples	Total samples	Minority class samples	Total samples	Minority class samples	Total samples
	Weak scaling (1)		Weak scaling (2)		Weak scaling (3)	
1	250	250000	250	2000000	2000	250000
2	500	500000	500	2000000	2000	500000
4	1000	1000000	1000	2000000	2000	1000000
8	2000	2000000	2000	2000000	2000	2000000
16	4000	4000000	4000	2000000	2000	4000000
32	8000	8000000	8000	2000000	2000	8000000
64	16000	16000000	16000	2000000	2000	16000000
128	32000	32000000	32000	2000000	2000	32000000

Figure 3.10: Summary of the datasets used for evaluating weak scaling of *parSMURF-NG*.

We evaluated the algorithm on the two HPC systems, and the results are reported in Figure 3.11 and show a remarkable speed-up (up to 91x on Marconi KNL and 100x on Hawk) at a very high efficiency. Although the speed-up between the two systems is comparable, Hawk runs apparently take considerably more time: this is due to the lower number of cores-per-node of the machines in this system - actually, the number of threads-per-node, as XeonPhi processors are able to schedule 4 threads per computing cores, compared to 2 of the AMD Rome architecture. This experiment fully fulfills our expectations in terms of scalability and efficiency; moreover, results are impressive on both architectures and are obtained by simply recompiling the code. Also, results show that *parSMURF-NG* is more than capable of dealing with Big Data, as memory consumption is well-optimized and should be capable of processing larger dataset.

Weak scaling

Real-world genetic problem designed to be solved with *parSMURF-NG* are always fixed-sized, thus it is very unlikely that a variable-size dataset needs to be processed with this software. Also, weak scalability is harder to measure for *parSMURF* because the computational complexity of the problem strongly depends on the number of samples assigned to the minority class, rather than the total number of samples of the dataset. For these reasons we believe that tests performed under the strong scalability setup provide a more fair and impartial analysis of the behavior of the algorithm. However, to prove that results are only dependent on the number of minority class samples, we designed three experimental setups differing only in the dataset



	N of MPI processes	Time to solution (s)	Ideal time to solution (s)	Speed-up	Efficiency
Cineca Marconi	1	37439.7	37439.70	-	100%
	2	18726.2	18719.85	1.999	100%
	4	11300	9359.93	3.313	83%
	8	7525.62	4679.96	4.975	62%
	16	3843.57	2339.98	9.741	61%
	32	1395.35	1169.99	26.832	84%
	64	769.277	585.00	48.669	76%
	128	408.99	292.50	91.542	72%
HLSR Apollo HAWK	1	175526	175526	-	100%
	2	92015	87763.00	1.908	95%
	4	55509.9	43881.50	3.162	79%
	8	36569.9	21940.75	4.800	60%
	16	18440.9	10970.38	9.518	59%
	32	6246.51	5485.19	28.100	88%
	64	3194.83	2742.59	54.941	86%
	128	1735.98	1371.30	101.111	79%

Figure 3.11: Strong scaling results of the Hawk system (green) and Cineca Marconi KNL partition (blue). Speed-up (left), efficiency (right) and actual times (bottom) of *parSMURF-NG* using a fixed size problem (strong scaling). In both graphs, x-axis shows the number of computing nodes. Efficiency shown in percentage. Each rank was launched with the following configuration: 8 threads for partition processing, 16 threads for random forest model generation and test, so the total number of threads for each rank is 128. *parSMURF-NG* hyper-parameters used on this test: nParts=256, fp=1, ratio=1, k=5, nTrees=50, mtry="default". Dataset is composed by 30000000 samples (2000 positives), each one having 100 features. *parSMURF-NG* was configured to execute a 10-fold cross-validation on the dataset.

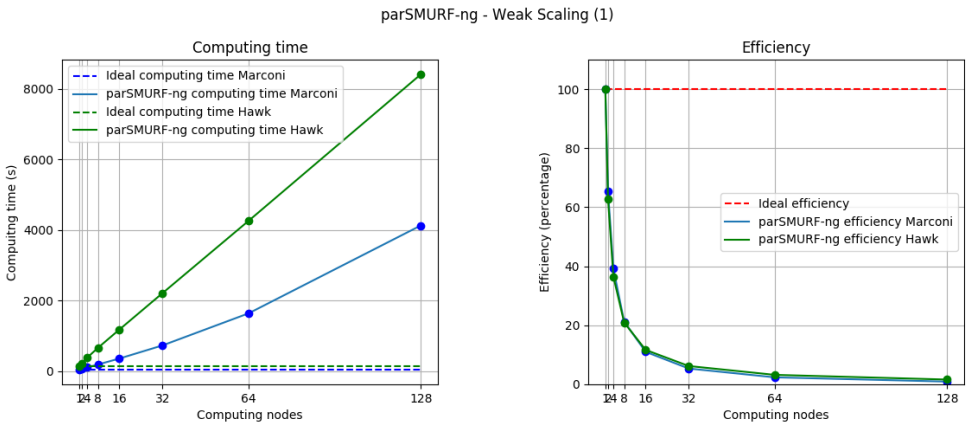
composition (see Figure 3.10). In the first run, we used a series of datasets whose size and number of positive samples increase proportionally with the number of computing nodes. Results are shown in Figure 3.12. Efficiency rapidly decreases, but results are misleading, since the same curve (Figure 3) can be obtained with a different experimental setup. Curves in Figure 3.13 have been obtained just by changing the distribution of samples: in this experiment we keep the datasize fixed to 2000000 samples, but we progressively increase the number of samples belonging to the minority class, in proportion with the number of assigned computing nodes (see Table 1). Computing time and efficiency are comparable to the previous experiment. Finally, we devised a third experimental setup for which the total number of samples increases as in the first experiment, but the number of samples belonging to the minority class is fixed (2000 samples). Results are reported in Figure 3.14, and again are very misleading: dataset composition follows the gold standard rule for the evaluation of weak scalability, however results show a superscalar behavior.

These contradicting results may be caused by different reasons. From the table in Figure 3.10 and the graphs in Figures 3.12, 3.13 and 3.14 we see a clear indication that the computational complexity is correlated with the number of examples in the minority class. This is justified by the fact that the partition size is ultimately set by the number of examples in the minority class assigned to each partition. Another reason is that the workload does not proportionally increase with the partition size, as more burden is placed in the training of the base learners: this could be addressed by increasing the number of computing threads assigned to the random forest training, but we have not explored this possibility in this specific testing phase.

3.3 Conclusions

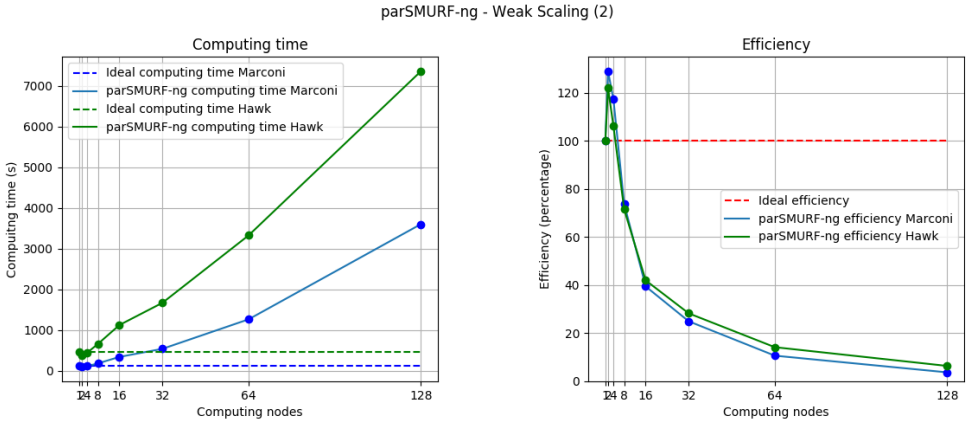
In this Chapter we presented *parSMURF*, a High Performance Computing tool for the prediction of pathogenic variants, designed to deal with the issues related to the inference of accurate predictions with highly unbalanced datasets in a genomic medicine context. We showed that *hyperSMURF*, despite its encouraging results with different genomic data sets, suffers from two major drawbacks: a very demanding computing time and the need of a proper fine tuning of the learning parameters. The proposed *parSMURF* method provides a solution for both problems, through two efficient parallel implementations - $parSMURF^1$ and $parSMURF^n$ - that scale well with respectively multi-core machines and multi-node HPC cluster environments.

Results with synthetic datasets show that *parSMURF* scales nicely with



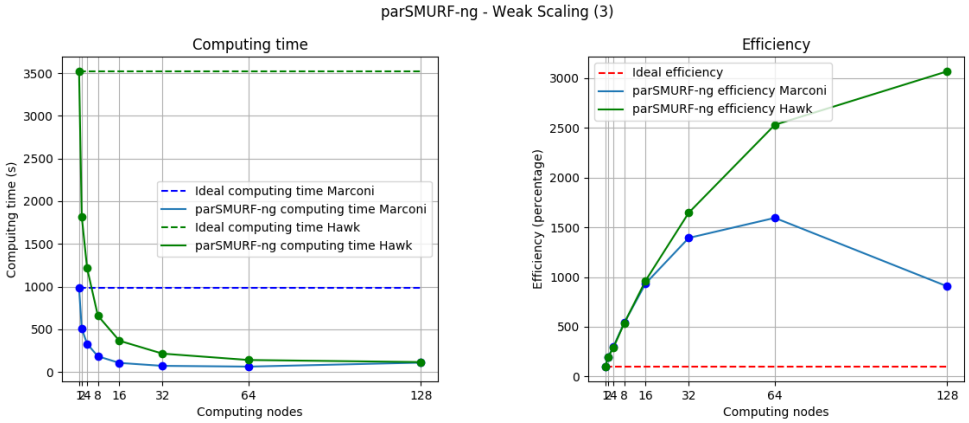
	N of MPI processes	Time to solution	Ideal time to solution (s)	Speed-up	Efficiency
Cineca Marconi	1	38.89	38.89	-	100%
	2	59.56	38.89	0.65	65%
	4	99.25	38.89	0.39	39%
	8	183.22	38.89	0.21	21%
	16	353.30	38.89	0.11	11%
	32	724.99	38.89	0.05	5%
	64	1635.40	38.89	0.02	2%
	128	4128.20	38.89	0.01	1%
HLSR Apollo HAWK	1	137.267	137.27	-	100%
	2	218.637	137.27	0.63	63%
	4	376.456	137.27	0.36	36%
	8	654.957	137.27	0.21	21%
	16	1175.93	137.27	0.12	12%
	32	2207.75	137.27	0.06	6%
	64	4260.36	137.27	0.03	3%
	128	8416.62	137.27	0.02	2%

Figure 3.12: Weak scaling results (setup 1) of the Hawk system (green) and Cineca Marconi KNL partition (blue). Computing time (left), efficiency (right) and actual times (bottom) of *parSMURF-NG* using a variable size problem with proportional increment of both positive and negative samples (weak scaling). In both graphs, x-axis shows the number of computing nodes. Computing time in seconds. Efficiency shown in percentage. Launch configuration and hyper-parameters as per Figure 3.11. Data size is proportional to the number of computing nodes: 1 node / 250000 samples (250 positives), 2 nodes / 500000 samples (500 positives), up to 128 nodes / 32000000 samples (32000 positives). Each sample has 50 features. *parSMURF-NG* was configured to evaluate only one fold out of a 10-fold cross-validation on the dataset.



	N of MPI processes	Time to solution	Ideal time to solution (s)	Speed-up	Efficiency
Cineca Marconi	1	135.19	135.19	-	100%
	2	104.87	135.19	1.29	129%
	4	115.16	135.19	1.17	117%
	8	183.27	135.19	0.74	74%
	16	342.59	135.19	0.39	39%
	32	541.43	135.19	0.25	25%
	64	1265.00	135.19	0.11	11%
	128	3599.47	135.19	0.04	4%
HLSR Apollo HAWK	1	472.39	472.39	-	100%
	2	387.507	472.39	1.22	122%
	4	444.635	472.39	1.06	106%
	8	660.345	472.39	0.72	72%
	16	1125.36	472.39	0.42	42%
	32	1670.94	472.39	0.28	28%
	64	3324.76	472.39	0.14	14%
	128	7350.37	472.39	0.06	6%

Figure 3.13: Weak scaling results (setup 2) of the Hawk system (green) and Cineca Marconi KNL partition (blue). Computing time (left), efficiency (right) and actual times (bottom) of *parSMURF-NG* using a variable size problem with proportional increment of both positive and negative samples (weak scaling). In both graphs, x-axis shows the number of computing nodes. Computing time in seconds. Efficiency shown in percentage. Launch configuration and hyper-parameters as per Figure 3.11. Data size varies differently with regard to Figure 3.12. The total number of samples is fixed (2000000), but the number of positive in each run varies proportionally to the number of computing nodes: 1 node / 2000000 samples (250 positives), 2 nodes / 2000000 samples (500 positives), up to 128 nodes / 2000000 (32000 positives). Each sample has 50 features. *parSMURF-NG* was configured to evaluate only one fold out of a 10-fold cross-validation on the dataset.



	N of MPI processes	Time to solution	Ideal time to solution (s)	Speed-up	Efficiency
Cineca Marconi	1	984.14	984.14	-	100%
	2	506.37	984.14	1.94	194%
	4	328.69	984.14	2.99	299%
	8	182.55	984.14	5.39	539%
	16	105.10	984.14	9.36	936%
	32	70.64	984.14	13.93	1393%
	64	61.72	984.14	15.95	1595%
	128	108.48	984.14	9.07	907%
HLSR Apollo HAWK	1	3521.54	3521.54	-	100%
	2	1822	3521.54	1.933	193%
	4	1213.44	3521.54	2.902	290%
	8	658.499	3521.54	5.348	535%
	16	367.776	3521.54	9.575	958%
	32	213.903	3521.54	16.463	1646%
	64	139.202	3521.54	25.298	2530%
	128	114.828	3521.54	30.668	3067%

Figure 3.14: Weak scaling results (setup 3) of the Hawk system (green) and Cineca Marconi KNL partition (blue). Computing time (left), efficiency (right) and actual times (bottom) of *parSMURF-NG* using a variable size problem with proportional increment of both positive and negative samples (weak scaling). In both graphs, x-axis shows the number of computing nodes. Computing time in seconds. Efficiency shown in percentage. Launch configuration and hyper-parameters as per Figure 3.11. Data size varies differently with regard to Fig. 3.12 and 3.13. The number of positive samples is fixed (2000), but the number of total samples in each run varies proportionally to the number of computing nodes: 1 node / 250000 samples (2000 positives), 2 nodes / 500000 samples (2000 positives), up to 128 nodes / 32000000 (2000 positives). Each sample has 50 features. *parSMURF-NG* was configured to evaluate only one fold out of a 10-fold cross-validation on the dataset.

large data sets, introducing a sensible speed-up with respect to the pure sequential version. Especially for large data sets, as expected, we should prefer the hybrid MPI-multi-thread version *parSMURF*^{n}, while for relatively smaller datasets we can obtain a reasonable speed-up also with the pure multi-thread version *parSMURF*^{1} that can run also with a off-the-shelf laptop or desktop computer, by exploiting the multi-core architecture of modern computers.

parSMURF features two different and both effective strategies for the auto-tuning of the learning parameters: the first is based on an exhaustive grid search which proves to be effective in finding the best combination of hyper-parameters in terms of maximizing the AUPRC rating, but turns out to be very computing intensive. The other strategy is Bayesian optimization based and aims to find a near-optimal hyper-parameter combination in a fraction of time compared to the grid search strategy. Experimental results with Mendelian diseases and GWAS hits in non-coding regulatory regions show that *parSMURF* can enhance *hyperSMURF* performance, confirming that fine-tuning of learning hyper-parameters may lead to significant improvements of the results.

The high level of parallelism of *parSMURF*, its autotuning hyper-parameters capabilities and its easy-to-use software interface allow the user to apply this tool to ranking and classification problems characterized by highly imbalanced big data. This situation commonly rises up in Genomic Medicine problems, since only a small set of "positive" examples is usually available to train the learning machines. For this reason *parSMURF* can be a useful tool not only for the prediction of pathogenic variants, but also for any imbalanced ranking and classification problem in Genomic Medicine, provided that suitable big data are available for the problem at hand.

We also discussed a currently in progress extension of the work which will be used for the evaluation of a new set of ReMM scores with the aid of the newly developed *parSMURF-NG*: this new version overcomes a few minor limitations of *parSMURF* and thanks to its improved scalability it represents the current state-of-the-art solution for dealing with highly imbalanced dataset in the context of Genomic Medicine. The proposal has been awarded by the PRACE consortium with 50 millions core-hours in the Leibniz Supercomputing Centre in Garching (Munich, Germany). The newly evaluated ReMM scores will be made available through the new version of Genomiser, the state-of-the-art tool for the molecular diagnosis of Mendelian diseases, hence expecting to achieve a major breakthrough in this field and proposing a major contribution to the scientific community.

Chapter 4

GPU-Based Deep Neural Networks for the tissue-specific prediction of active regulatory regions in the human genome.

The annotation and characterization of tissue-specific cis-regulatory elements (CREs) in non-coding DNA represents an open challenge in computational genomics. Several prior works show that machine learning methods, using epigenetic or spectral features directly extracted from DNA sequences, can predict active promoters and enhancers in specific tissues or cell lines. In particular, very recently deep-learning techniques obtained state-of-the-art results in this challenging computational task.

In this study, we provide additional evidence that Feed Forward Neural Networks (FFNN) trained on epigenetic data and one-dimensional convolutional neural networks (CNN) trained on DNA sequence data can successfully predict active regulatory regions in different cell lines. We show that model selection by means of Bayesian optimization applied to both FFNN and CNN models can significantly improve deep neural network performance, by automatically finding models that best fit the data.

Further, we show that techniques applied to balance active and non-active regulatory regions in the human genome in training and test data may lead to over-optimistic or poor predictions. We recommend to use actual imbalanced data that was not used to train the models for evaluating their generalization performance.

4.1 Background

Non-coding DNA regions, which include 98% of the human genome, are that part of DNA that does not encode for structural proteins and enzymes. A subset of those regions, so-called cis-regulatory elements (CREs) determine spatiotemporal patterns of gene expression [270, 271] and therefore play a key role in the control of transcription. CREs are involved in the development of different cell types/tissues, in the timing and intensity of gene expression during the cell life, in the dynamical response to changes in physiological conditions through interactions with DNA-binding transcription factors (TFs), and in the focal alteration of chromatin structure [272]. Genome-wide association studies (GWAS) discovered thousands of variants associated with diseases and traits enriched in non-conding sequences, and several lines of research show that genetic variants in regulatory regions may be deleterious or directly involved in genetic diseases [109, 99, 235].

A great deal of research has been devoted to the identification of CREs and to their cell-specific activation status. Such studies are essential to dissect the mechanisms underlying the modulation of gene expression and to understand the functional impact of genetic variants on human diseases. Indeed, the effect of genetic variants in non-coding regions is strongly related to the prediction of active regulatory regions (e.g. nucleosome-free regions that are accessible by TFs). Conversely, if a genetic variant, even if potentially deleterious/functionally constrained (e.g. high conservation), is located in an inactive DNA region, it is less likely to be pathogenic.

Thus, great effort has been undertaken to map TF binding sites and histone modifications across cell types and tissues [273, 274, 25, 275, 276]. In particular, the ENCODE project [25] identified promoters and enhancers in 147 cell types using a wide range of high-throughput technologies, while the FANTOM project employed CAGE (Cap Analysis of Gene Expression) technologies to broaden the spectrum of considered samples, including 1,816 human and 1,016 mouse samples [26, 277]. The Roadmap Epigenomics Consortium [278] studied the epigenomic landscape of 111 representative primary human tissues and cell-lines. However, the experimental identification of CREs is still expensive and time consuming, and, despite the efforts of the above-mentioned projects, researches are still far from obtaining a comprehensive mapping of CREs across all cell types, disease status and developmental stages.

The use of computational methods, and in particular machine learning approaches, can be a crucial tool to identify the location and activation status of these regions. To this aim, initial approaches, due to the reduced set of reliable annotations, applied unsupervised learning techniques [122, 123] to

data from the ENCODE project [25]. However, their low accuracy (around 26%) in predicting enhancer [279], pushed the development of more sophisticated supervised learning models, such as random-forest methods [124] and AdaBoost-based models [127]. The subsequent availability of large-scale and high-resolution CREs provided by the FANTOM5 Consortium [26] enabled the development of ensembles of support vector machines [126], and allowed for more complex models, such as deep neural networks[136, 140].

Deep learning models significantly contributed to the advance of machine intelligence [280, 281, 282], providing the ability to model complex systems and capture high-level patterns from data, when an underlying, even non-obvious, structure is present.

Recently, deep learning methods have been applied in regulatory genomics [283] and, in this context, the identification of CREs shows promising results. In particular, DeepEnhancer [136] uses CNNs to identify cell-specific enhancers from the genomic background using only sequence data; BiRen [135] similarly uses a hybrid deep learning architecture that integrates a gated recurrent unit-based bidirectional recurrent neural network and a CNN to predict human and mouse enhancers from sequence data.

Considering that approaches that employ only DNA sequences do not take into account the regulatory mechanisms encoded in the epigenetic data, PEDLA [133] predicts enhancers by using an extensive set of heterogeneous data (i.e. epigenomic, sequence, and conservation data) and a novel hybrid architecture integrating a deep neural network and a hidden Markov model. Interestingly, PEDLA iteratively learns from 22 training cell types/tissues and achieves high accuracy when predicting across 20 independent test cell types/tissues, showing high and consistent generalization performance across samples

To improve the prediction capabilities of these methods by explicitly taking into account whether CREs are active in the considered cell-lines/tissues, DECRES [140] uses FFNN to identify not only the presence of enhancers and promoters, but also if they are active in a specific human cell-line. To this end, authors extracted annotation data from FANTOM [284] and a wide set of epigenomic features from ENCODE [25].

This method is able to predict active enhancers and promoters from non-active regulatory regions, achieving high performance in all the considered tasks and outperforming state-of-the-art unsupervised methods. Moreover, DECRES predictions extend the FANTOM enhancer atlas of 16,988 bidirectionally transcribed loci, therefore providing the most complete annotation of CREs in the human genome so far. Nevertheless, DECRES exploits an experimental setup where both the training and the test sets are balanced, thus distorting the actual distribution of the data. The training set is typically

balanced to avoid the creation of a model which overfits the over-represented class, but the test set is kept unbalanced to avoid biasing the performance estimation. The peculiar balancing scheme adopted by the authors of DECRES poses the question how train and/or test set balancing affects classifier performance.

In this study, we show results obtained for predicting the activity state of CREs. We developed two deep neural network models (a Feed Forward Neural Network (FFNN) and a Convolutional Neural Network (CNN)) and investigated how the chosen model and the experimental setup (i.e. the balancing of the training and the test set) influence performance. We apply Bayesian optimization for model selection, and we analyze the impact of different balancing techniques on the overall results and performance evaluation.

Our results show that: 1) model selection by Bayesian optimization has the potential of improving performance; 2) CNN trained only on sequence data are competitive with FFNN trained on combined and richer epigenetic features, thus suggesting that multi-modal architectures can be applied in this context; 3) experimental setup plays a central role in the evaluation of the performance – we show that naive balancing techniques may lead to over-optimistic evaluation of the results and to poor training results.

4.2 Data set

Machine learning algorithms were trained and tested on genomic (DNA) regions of transcriptionally active enhancer and promoters downloaded from FANTOM5 [26] and matched features collected by Yifeng Li et. al [140] from ENCODE [25] for four cell lines, as follows: (GM12878, HelaS3, HepG2, K562).

- **GM12878**: a lymphoblastoid cell line produced from the blood of a female donor with northern and western European ancestry by EBV transformation.
- **HelaS3**: an immortalized cell line that was derived from a cervical cancer patient.
- **HepG2**: a cell line derived from a male patient with liver carcinoma.
- **K562**: an immortalized cell line produced from a female patient with chronic myelogenous leukaemia (CML).

Yifeng Li et. al [140] defines six different classes, active enhancers (A-E), active promoters (A-P), active exons (A-X) and their inactive counterparts (I-E, I-P, I-X), using thresholds on tags per million (TPM) values of the Cap Analysis of Gene Expression (CAGE) data set downloaded from the FANTOM5 database[1]. Classes of active and inactive exons are defined based on exon transcription levels from RNA-seq data downloaded from ENCODE[2]. Finally, an unknown class (UK) labels regions sampled from the genome, but excluding those regions overlapping FANTOM CAGE tags, exons and DNaseI peaks. The employed genomic regions and thresholds, as well as the considered features, are the same as those used in [140]. Table 4.1 provides an overview of the class distribution in each cell line.

Table 4.1: For each data set (on columns) we report the number of samples per class and the cardinality of the whole data set.

Labels	**HepG2**	**HelaS3**	**K562**	**GM12878**
A-E	1465	1847	894	2878
A-P	11467	10759	10076	10816
A-X	9931	9123	9033	8226
I-E	34556	32179	34392	28156
I-P	96184	79004	82829	73891
I-X	19071	22071	20261	19078
UK	79417	81502	78081	80004
Total	252091	236485	235566	223049

To train and test FFNN methods, the feature set consisted of histone modification and TF binding ChIP-seq, DNase-seq, FAIRE-seq, and ChIA-PET data from ENCODE [25] CpG islands and phastCons scores were included in the feature set by computing the mean value of the feature signal which falls within 200-bps bins centered at each labelled region.

To train and test CNN methods, we used sequence data obtained from the human reference genome GRCh37/hg19 In particular, each genomic region is represented by a sequence of 200 one-hot encoded nucleotides.

Thus, for each cis-regulatory element (regardless of labelling), we have two different representations: (1) a set of numeric features suitable for training FFNN models [285], and (2) nucleotide sequences to be processed with CNN models [286].

[1]FANTOM5 Data at `http://fantom.gsc.riken.jp/5/data`.

[2]ENCODE Data at `ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC`.

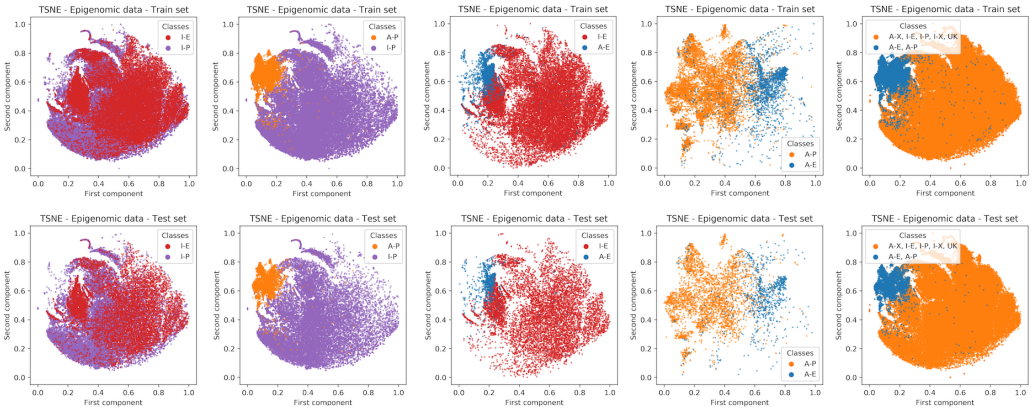


Figure 4.1: Decomposition in the first two principal components of epigenomic data (cell line GM12878) using TSNE for 1 of the 10 generated hold-outs. Top row: train set; bottom row: test set. Each column shows a different classification task. In order, from left to right: Inactive Enhancers vs Inactive Promoters, Active Promoters vs Inactive Promoters, Active Enhancers vs Inactive Enhancers, Active Enhancers vs Active Promoters, Active Enhancers and Promoters vs anything else.

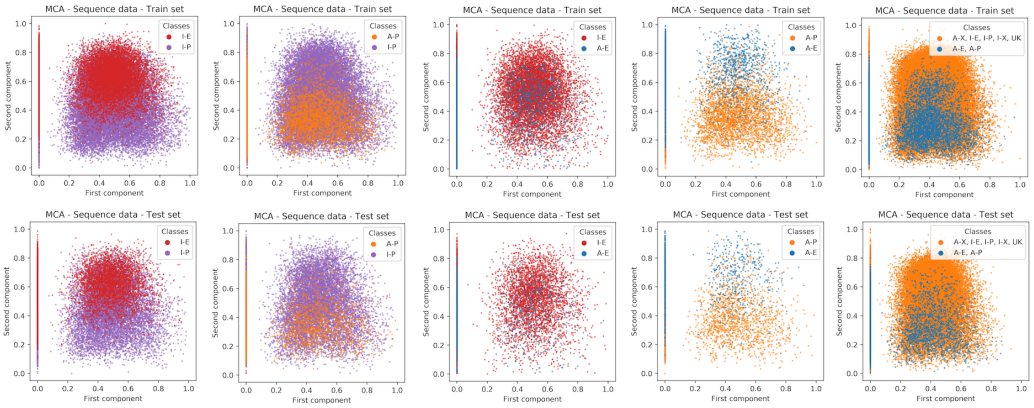


Figure 4.2: Decomposition in the first two principal components of sequence data (cell line GM12878) using MCA for 1 of the 10 generated hold-outs. Top row: train set; bottom row: test set. Each column shows a different classification task. In order, from left to right: Inactive Enhancers vs Inactive Promoters, Active Promoters vs Inactive Promoters, Active Enhancers vs Inactive Enhancers, Active Enhancers vs Active Promoters, Active Enhancers and Promoters vs anything else.

To provide some insights into the class distribution of the data, we show the projections of the training matrices for one of the cell lines (GM12878) with the two principal components computed by t-SNE [287] for the epigenomic data set (figure 4.1) and MCA [288] for sequence data (figure 4.2). MCA and t-SNE plots for tasks "AP vs IP", "AE vs AP" and "AE+AP vs ELSE" (second, fourth and fifth columns in fig. 4.1 and 4.2, respectively) show that the classes are well separated. Hence, we expect that our models will reach good performance in such classification tasks. MCA decomposition for task "AE vs IE" (third column) shows no clear pattern, while t-SNE shows a clearly separable task. Here, we expect that the CNN model will perform worse than FFNN. Finally, an high level of entanglement among classes is shown in the t-SNE decomposition for task "IE vs IP" (first column), therefore we expect that CNNs will provide better predictions for this task.

In the following we describe the characteristics of each model and the employed bayesian optimization technique. All the presented models were developed using Keras [38] with the TensorFlow backend [39].

4.3 Methods

The identification of active regulatory regions can be modeled as a binary classification task. To perform our experiments, we used Feed Forward Neural Networks (FFNN, [285]) for processing epigenomic data, while sequence data was analyzed using Convolutional Neural Networks (CNN, [286]).

For each method (FFNN and CNN) we developed a "fixed" baseline model, and an "optimized" model, with hyper-parameters selected by Bayesian optimization Therefore, we obtained four different models, which we call *fixed-FFNN*, *fixed-CNN*, *Bayesian-FFNN* and *Bayesian-CNN*.

4.3.1 Bayesian Optimization

When designing a neural network (NN) for a given classification task, the choice of architecture (number of layers, neurons and activation functions) and setting of learning hyper-parameters (e.g. optimizer algorithm, batch size, learning rate) are critical for achieving reliable and high performances.

At current time state-of-the-art, no well-accepted or unified method for manually or automatically finding the appropriate hyper-parameters for a given task has been defined, and model selection is generally performed by relying on past experience, involving empirical tests or applying automatic methods which explore the hyper-parameter space in a bounded domain.

"Grid search" is one such approach which finds the best combination of learning hyper-parameters and network architecture by exhaustively evaluating a user-defined objective function for every possible hyper-parameter combination in the bounded domain of the search space, looking which combination realizes the absolute minimum/maximum of such objective function. Although being effective and highly parallelizable, grid search suffers from its high computational costs, exponentially increasing with the dimension of the hyper-parameter space. For this reason, alternative approaches have been proposed in recent years. For instance, "random search" [289] efficiently explores the hyper-parameter space by evaluating a sequence of randomly extracted points, while "Spectral approach" [290] applies the Fourier transform to search the maximum or minimum of the objective function in the frequency domain.

"Bayesian Optimization" (BO) [291] has proven to be an effective and cost efficient solution to hyper-parameter optimization. Briefly, the idea is that the "objective function", characterized by high cost for the evaluation of each point of a bounded domain, can be approximated by building a probabilistic model (the "surrogate function") which is relatively cheaper to query. Optimization can then be performed by substituting the objective with the surrogate function. As the surrogate function represents an "a priori" distribution of the objective function, given some observations obtained by evaluation of the objective, it is possible to exploit Bayes's rule to generate an "a posteriori" estimation of the (objective) function and then update the probabilistic model (surrogate function). Candidates for the observations are suggested through an appropriate "Acquisition function" which uses the information gained by the probabilistic model (estimated by the n already observed points) for suggesting the next $n+1$ candidate.

Depending on the task, it is possible to select the most appropriate acquisition function from a wide array of choices. A common trait of these functions is that they all act upon the criteria of "exploration versus exploitation", so that the sequence of suggested points will provide a better overlook of the objective function (exploration) or a better identification of its maximum/minimum (exploitation). A comprehensive review of possible acquisition functions is available at [292].

4.3.2 Fixed-FFNN and Bayesian-FFNN

We designed a baseline *fixed-FFNN* whose architecture is composed of three cascading fully-connected layers (with 16, 4 and 2 neurons, respectively) with the Rectified Linear Unit (ReLU) [293] activation function. A final layer structured as a single neuron with sigmoid activation function

acts as output layer, computing the final binary predictions. During network training, weight values were adjusted using the Stochastic Gradient Descent iterative method with a relatively small value for the batch size hyper-parameter (32), learning rate 0.5, learning rate decay of 0.1, l_2 regularizer of 0.0 and no momentum; the net get trainied for a maximum of 64 epochs.

This baseline model acted as a reference for developing the *Bayesian-FFNN*, featuring automatic model selection. To perform this task, in [140] authors used a grid search, which exhaustively explores the hyper-parameter space of the folloing variables: the network configuration, i.e. the number of hidden layers (from 0 to 3) and the number of their units (from 0 to a maximum of 256, 128, 64 neurons in the first, second and third layer, respectively), the initial learning rate, and the l_2-regularization amount in continuous intervals (not reported by the authors).

Our setup differs from that in [140] since we substituted the computationally expensive grid search with Bayesian optimization [255], which maximizes the mean AUPRC computed over the validation sets of 10 internal hold-outs). More precisely, Bayesian optimization chooses the network architecture from the previous search space [140], while keeping other hyper-parameters equal to those of *fixed-FFNN*. Once the optimal architecture was found, we optimize other learning hyper-parameters. The *fixed-FFNN* and *Bayesian-FFNN* models are summarized in table 4.2. In particular, we note that the chosen number of layers in *Bayesian-FFNN* was often equal to that of *fixed-FFNN*, the number of chosen units was always bigger than that of *fixed-FFNN*, and the learning parameters were often selected in the lower spectrum of the continuous search interval, except for the maximum number of epochs.

4.3.3 Fixed-CNN and Bayesian-CNN

To analyze raw DNA sequence data, we used 1D-Convolutional Neural Networks. Like for FFNN models, we first developed a fixed model to assess whether this approach effectively recognizes active regulatory regions. After obtaining promising results, we aimed at improving performance by automatically selecting the CNN model parameters through Bayesian optimization.

The *fixed-CNN* model is outlined in Table 4.3. The core of the network is composed of three consecutive blocks, each consisting of three (consecutive) convolutional layers followed by one 1-dimensional max/average pooling layer. The number of units in the three convolutional layers of each block, as well as the filter sizes, are fixed. A filter size of 5 for the first three convolutional layers was chosen as this represents a reasonable motif size. As for

Table 4.2: The architecture and learning parameters of the *fixed-FFNN* (top table) and the *Bayesian-FFNN* (bottom table) models. Optimized hyper-parameters are highlighted with coloured cells. The hyper-parameter space explored by the bayesian optimizer is shown in square brackets (continuous bounded interval) or curly brackets (finite discrete set). Selected values are highlighted with bold characters.

fixed-FFNN		
Layers	**Units**	**Activation**
Fully connected	16	ReLU
Fully connected	4	ReLU
Fully connected	2	ReLU
Output	1	Sigmoid

Learning parameters	
learning rate	0.5
learning rate decay	0.1
l2 regularizer	0.0
batch size	32
weight value estimation	SGD
max no. epochs	64

Bayesian-FFNN		
Layers	**Hyper-parameter Space**	**Activation**
No. of fully connected layers	{0, 1, 2, **3**}	
No. of units layer 1	{**256**, 128, 64, 32, 16, 8, 4, 2}	ReLU
No. of units layer 2	{**128**, 64, 32, 16, 8, 4, 2}	ReLU
No. of units layer 3	{**64**, 32, 16, 8, 4, 2}	ReLU
Output	1	Sigmoid

Learning parameters	
learning rate	[**0.1**, 0.5]
learning rate decay	[**0.01**, 0.2]
l2 regularizer	[0, .., **0.001**, .., 0.1]
batch size	[32, .., **100**, .., 256]
weight value estimation	SGD
max no. epochs	[32, **1000**]

the FFNN models, all neurons in each layer have ReLU activation function with the exception of the output layer, where the output neuron has sigmoid activation. The Nadam algorithm was used to adjust weight values during training, learning rate was set to 0.002, and batch size set to 100 examples.

Considering that the *fixed-CNN* architecture has many weights that need to be set, we applied Bayesian optimization to simplify its architecture. We maximized the mean AUPRC computed over the validation sets of 10 internal hold-outs to choose the number of blocks from 1 to 3, and to choose, for each layer, a number of units lower than that of the *fixed-CNN* model. In table 4.4, the architecture of the *Bayesian-CNN* model is shown. Again, the Nadam algorithm was used to estimate the weight values, the learning rate is set to 0.002, and the batch size to 100.

Table 4.3: Architecture of the *fixed-CNN* model. "Max Pool 1D" refers to max-pooling 1D layer, "Avg. Pool 1D" refers to average-pooling 1D layer and "Batch Norm" refers to a batch normalization layer.

fixed-CNN					
Layers	**Type**	**Units**	**Kernel**	**Activation**	**Notes**
3	Convolutional	64	5	ReLU	-
1	Max Pooling 1D	-	-	-	size 2
3	Convolutional	128	3	ReLU	-
1	Max Pooling 1D	-	-	-	size 2
3	Convolutional	128	3	ReLU	-
1	Avg. Pooling 1D	-	-	-	-
1	Dropout	-	-	-	Prob. 0.5
2	Dense	10	-	ReLU	-
1	Dropout	-	-	-	Prob. 0.5
1	Dense	1	-	Sigmoid	-

Learning parameters	
learning rate	0.002
batch size	100
weight value estimation	Nadam
epochs	100

4.3.4 DeepEnhancer

We compared our *Bayesian-CNN* model with the five DeepEnhancer networks [136], which are 1D-CNNs using one-hot encoded sequence data for distinguishing enhancers from background sequences.

Table 4.4: Architecture of the *Bayesian-CNN* model. Values optimized by BO are highlighted with coloured cells. Square brackets show the explored hyper-parameter space; bold value font indicates the selected values. "Max Pool 1D" refers to max-pooling 1D layer, "Avg. Pool 1D" refers to average-pooling 1D layer and "Batch Norm" refers to a batch normalization layer.

Bayesian-CNN					
Layers	**Type**	**Units**	**Kernel**	**Activation**	**Notes**
3	Convolutional + Batch Norm	64	5	ReLU	-
1	Max Pooling 1D	-	-	-	Size 2
1	Convolutional + Batch Norm	{32, **64**, 128}	{5, **10**}	ReLU	-
1	Max Pooling 1D	-	-	-	Size 2
1	Flatten	-	-	-	-
1	Dense	{10, 32, **64**}	-	ReLU	-
1	Dropout	-	-	-	Prob. 0.1
1	Dense	{10, 32, **64**}	-	ReLU	-
1	Dropout	-	-	-	Prob. 0.1
1	Dense	1	-	Sigmoid	-

Learning parameters	
learning rate	0.002
batch size	100
weight value estimation	Nadam
epochs	100

For DeepEnhancer, the best performing model is 4conv2pool4norm [136] which consists of four convolutional layers, each one followed by a batch normalization operation. In the second and fourth layer, batch normalization is followed by a max-pooling layer. The first two convolutional layers contain 128 kernels of shape 1×8, while the other convolutional layers contain 64 kernels with shape 1×3. They are followed by two dense layers, with 256 and 128 neurons respectively, interleaved with a dropout layer (ratio 0.5), while a final 2-way softmax layer computes the classification probability results. The ReLU activation function is employed in the dense layers.

We used an implementation of this 4conv2pool4norm model to distinguish the activity state of enhancers versus promoters. We kept network structure and hyper-parameters of the original 4conv2pool4norm model (see [136] for further details), but modified the output layer by substituting the 2 output neurons (with softmax activation) with one single output neuron

with a sigmoid activation function to generate the final binary predictions, in line with our FFNN and CNN models.

4.3.5 GPU parallelization

As very briefly discussed in Section 4.2, we implemented the proposed approach in Python 3, leveraging the Tensorflow and Keras frameworks for the Deep Neural Network part of the code. In particular, Tensorflow is an highly optimized library specific for the development and deployment of DNNs: its computational core is able to seamless exploit CPU cores and GPU accelerators for providing almost state-of-the-art results in terms of computing time and performance. For this reason, we avoid to optimize the internal operation of the library and we focused on a more coarse grained task which represented a major bottleneck in computing performance, that is the selection of the best model by grid search and Bayesian optimization.

During the development of the CNN and FFNN networks, we noticed that training and testing of both networks are tasks characterized by high computational time. This is especially aggravated during model selection, as several models need to be trained and tested. However, we also noticed that training of the FFNN models requires a remarkable low GPU cores occupancy and low GPU memory occupancy (the opposite holds for the CNN models, as training and testing are high GPU and memory occupancy tasks), hence it is possible to train and test several models concurrently even on a single GPU without incurring in computational penalties.

Hence we parallellized the model selection phase, so that several models can be evaluated concurrently, speeding-up the computation. For this task we exploited the master/slave parallel paradigm: in our design, one master process is in charge of managing the inter-process communication, the collection of the results and the delivery of the hyper-parameter combinations to be evaluated (either by scanning a grid or being generated by a Bayesian optimizer), while a pool of slave processes do the actual model evaluation.

To this end we designed a system of message passing between the master and each slave processes. This inter-process communication scheme is summarized in Figure 4.3. As master and slaves processes do not share a common memory space, inter-communication between processes has been implemented via the message passing paradigm using the Queue Python class. Each message contains a status word and may also contain data. The master assigns an input Queue - in the Figure, *results_ queue* - and an output Queue - *parameters_ queue* - to each slave process. Output queues are used by the master process to provide instructions and data to each slave process, while input queues are used by slave processes to signal their status or return the

results of the evaluation of an hyper-parameter combination to the master process.

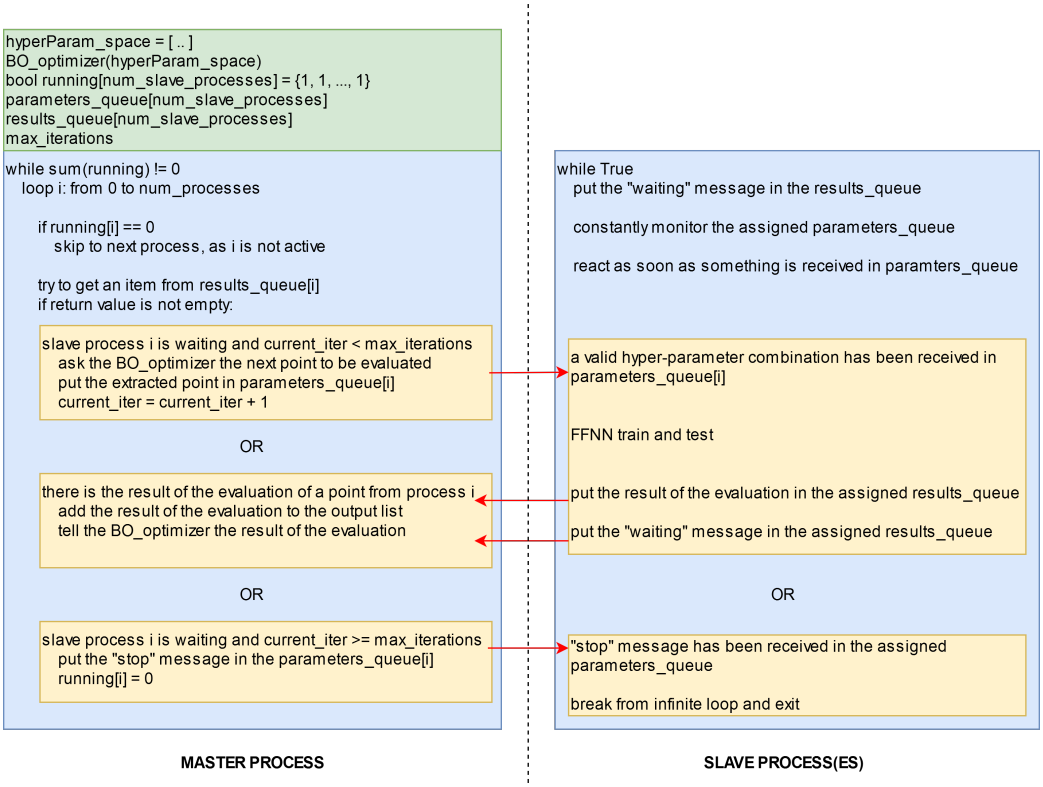


Figure 4.3: High level scheme of the inter-process communication protocol. On the left side, pseudo-code of the master process; right side, pseudo-code of each slave process. Message passing occurs vie Python Queue objects: red arrows indicate the direction of the message.

As a brief comment to the high level pseudo-code, the main process iterates on the slave processes, skipping the inactive ones and checking the assigned output queue. If the queue is not empty, one of the three following condition could be met:

- no stopping condition has been met, the slave process has put the "waiting" message in its input queue and is currently idle. Then, the master process proceed to interrogate the Bayesian optimizer, obtaining a new hyper-parameter combination to be evaluated; it sends the extracted point to the slave process by placing it in the output queue and increments the number of current evaluations.

- the result of the evaluation of an hyper-parameter combination has

been placed in the queue. The master process passes the result, along with the evaluated point, to the Bayesian optimizer. This point is used by the BO for updating its internal model. Note that after placing the result in the input queue, the slave process places also the "waiting" message and suspends its execution.

- one of the stopping condition has not been met, the slave process has put the "waiting" message in the corresponding input the queue and is currently idle. The master process sends the "stop" message to the slave and updates its status from "running" to "not running"

This loop ends once all the slave processes have stopped. The stopping conditions are two: model selection may end if the maximum number of evaluations (set by the variable *max_iterations*) has been reached, or if no significant performance improvement in the last 3 evaluated models is recorded.

On the other hand, each slave process constantly monitor its input queue. It can receive two types of messages: a valid hyper-parameter combination, and in this case it is evaluated by building, training and testing the corresponding model; otherwise, if the "stop" message is received, the slave process terminates itself.

Grid search works exactly in the same way, with the minor difference that points are not in-line generated, but are extracted from a list containing all the possible points of the pre-defined search space.

We want to remark that this task would have been more easily implemented using multi-threading instead of multi-processing, but Python is unsuitable for the former execution model, as Python interpreter includes a Global Interpreter Lock mechanism, so threads in a pool are executed sequentially in round robin at regular intervals: to preserve the status between objects and prevent race conditions, only one thread can be active in any moment.

4.4 Results and discussion

4.4.1 Experimental setup

For each of the cell lines introduced in the dataset Section, we test our methods on five dichotomic tasks proposed in [140]:

1. Inactive Enhancers vs Inactive Promoters (IE vs IP);
2. Active Promoters vs Inactive Promoters (AP vs IP);

3. Active Enhancers vs Inactive Enhancers (AE vs IE);
4. Active Enhancers vs Active Promoters (AE vs AP);
5. Active Enhancers and Promoters vs anything else ((AE + AP) vs ELSE);

We note that while the first four tasks use only samples belonging to the involved classes, task 5 requires using all the samples in the data set. This leads to classification tasks having different class imbalance, with ratios ranging from 2.5 to 38.5 (see table 4.5).

Table 4.5: Imbalance factor of different tasks and cell lines. Unbalanced setup (first to fifth column): for each data set and each task (first five rows from top to bottom), we report the class imbalance factor, measured as the ratio of cardinalities for the higher represented and lower represented class. The fifth column shows the average imbalance over all cell lines, when an unbalanced setup is used. Full-balanced setup (sixth column), imbalance factors for each task (equal across cell lines).

Unbalanced					
Task	**HepG2**	**HelaS3**	**K562**	**GM12878**	**Avg per task**
IE vs IP	2.78	2.46	2.41	2.62	2.57
AP vs IP	8.39	7.34	8.22	6.83	7.70
AE vs IE	23.59	17.42	38.47	9.78	22.32
AE vs AP	7.83	5.83	11.27	3.76	7.17
AE + AP vs else	18.49	17.76	20.47	15.29	18.00
Avg per cell line	12.22	10.16	16.17	7.66	11.55

Full-balanced [140]	
Task	**All cell lines**
IE vs IP	1
AP vs IP	2
AE vs IE	2
AE vs AP	1
AE + AP vs else	8
Avg per cell line	2.8

All classification tasks are executed using 10 randomly generated (external) hold-outs, each composed by splitting the data set into training (contain-

ing 70% of samples) and test set (containing 30% of samples). Classification tasks involving model selection were performed using additional 10 internal hold-outs (with the same proportion, 70%-30%, of train and test samples). The internal hold-outs are generated by randomly splitting each training set 10 times, thus forming 10 (internal) training and validation sets, used in Bayesian Optimization to select the best model by maximizing performance (AUPRC) on the validation sets. Training features are normalized using "MinMax" scaling between 0 and 1. The same normalization is applied on validation and test sets.

Performance is measured by using both Area Under the Receiver-Operating Curve (AUROC) [294] and Area Under the Precision-Recall Curve (AUPRC) [295] metrics computed over all test sets in the 10 hold-outs. While AUROC is a de-facto standard for evaluating classifier performance, AUPRC is more suitable when dealing with unbalanced data sets [296, 250, 164].

To investigate the effects of class balancing in training and test set data, all experiments were repeated three times and applied the following balancing setups:

1. full-balanced setup as proposed in [140]. Initially, the train and test set are randomly sampled to have 70% and 30% of samples in training and test sets, respectively. The training set is then randomly downsampled to at most 3000 samples per class. This provides a balanced training set. The samples in the training and test sets are also randomly downsampled to generate a corresponding test set with proportion of A-E:A-P:A-X:I-E:I-P:I-X:UK=1:1:1:2:2:1:10, according to the setup proposed in [140]. Note that the described test set subsampling greatly reduces class imbalance reported in Table 4.5 for all available cell lines.

2. balanced setup: only the training set is balanced as described in 1);

3. unbalanced setup: any balancing is avoided, maintaining the imbalance that characterizes the original class distribution (Tables 4.1 and 4.5).

4.4.2 Bayesian optimization improves prediction performance of FFNN and CNN models

Our first goal was to investigate the effect of model selection on generalization performance of FFNN and CNN models, to inspect whether a

systematic exploration of the hyper-parameter space leads to better classification results.

To this end, by using the aforementioned unbalanced setup, we assessed the performance of the two FFNN learning models (using epigenomic features) and the two CNN models (based only on sequence) for the five classification tasks listed in the experiental setup subsection. For each combination of model / task / data set, mean AUPRC and AUROC computed over the 10 repetitions and over the cell lines are shown in Figure 4.4. Performance of "fixed" and "Bayesian" models on each task were compared by applying Wilcoxon signed rank tests (at 0.01 significance level) [297, 298, 299] to detect statistically significant differences in the mean values of the classifiers AUPRC and AUROC distributions.

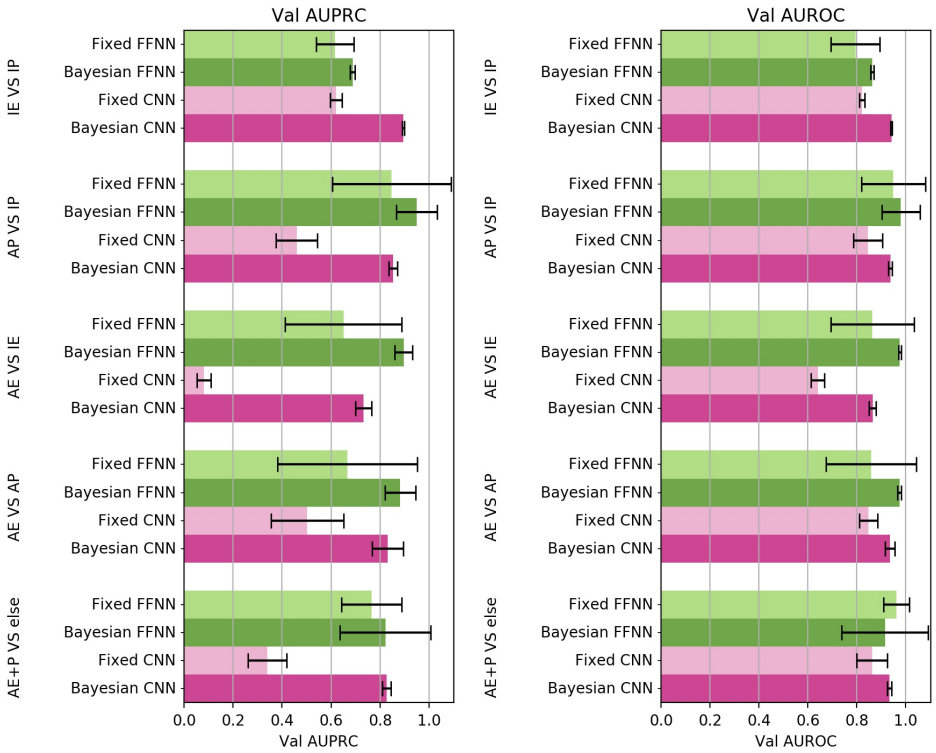


Figure 4.4: Comparison of deep neural network models with fixed parameters and Bayesian optimized parameters. The plotted AUPRC (left) and AUROC (right) are averaged over cell lines. Black bars represent standard deviations.

For AUPRC, Wilcoxon test showed a statistically significant difference between *Bayesian-FFNN* and *fixed-FFNN* in all tasks. We remark that *Bayesian-FFNN* achieved a better AUPRC value than *fixed-FFNN* for all

AUPRCs computed over all hold-outs, tasks, and cell lines, i.e. *Bayesian-FFNN* outperformed *fixed-FFNN* in 200 out of 200 comparisons.

Comparing performance of the two CNN models, *Bayesian-CNN* significantly outperformed *fixed-CNN*. Indeed, when looking again at the full list of AUPRC values across hold-outs, tasks, and cell lines, *Bayesian-CNN* always outperformed *fixed-CNN*. Our results suggest that model selection by Bayesian optimization improved AUPRC results.

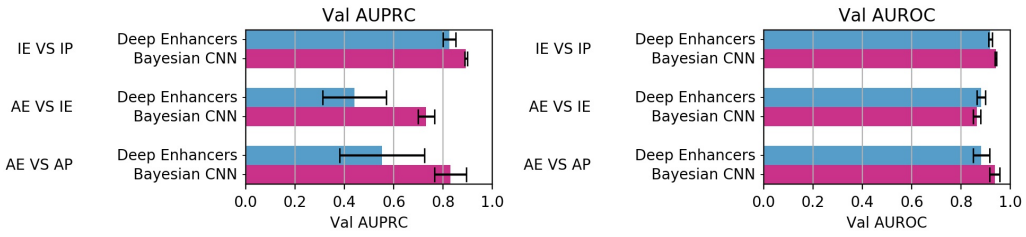
Regarding AUROC results, *Bayesian-FFNN* scores better average ratings than *fixed-FFNN* in all tasks with the exception of "AE+AP vs else". Also, Wilcoxon tests detected a statistically significant difference in mean between *Bayesian-FFNN* and *fixed-FFNN* in all tasks but one ("AE+AP vs else"). Considering all AUROC values computed over the 10 hold-outs, five tasks, and four cell lines, *Bayesian-FFNN* outperformed the *fixed-FFNN* the 93% of the times (186 out of 200).

Comparing CNN models, we note again that *Bayesian-CNN* always outperforms *fixed-CNN*. These results suggest that Bayesian model selection is a valid aid for improving both AUPRC and AUROC values of CNN models. This is also true for *Bayesian-FFNN*, as model selection improved the results in both AUPRC and AUROC for almost all tests.

4.4.3 Bayesian CNN models show comparable results with state-of-the-art methods

Due to the very promising performance achieved by *Bayesian-CNN*, we decided to further assess its capability for detecting active regulatory regions from raw DNA sequences by comparing it with the best performing DeepEnhancer neural network model (the 4conv2pool4norm net, see the DeepEnhancer section and [136]). Precisely, we used the unbalanced setup and the four cell lines to perform the three classification tasks involving enhancers ("IE vs IP", "AE vs IE" and "AE vs AP"). Using 4conv2pool4norm for these classification tasks is appropriate, as DeepEnhancer networks have been developed for recognizing enhancers against the background genome, and the original authors [136] state that it can be used for similar tasks.

Both models were assessed using the 10 hold-outs. In Figure 4.5 we show, for each of the three tasks, the mean AUPRC (left) and the mean AUROC (right). Wilcoxon tests confirmed that the difference in means of the computed AUPRC and AUROC distributions are statistically significant. Looking at individual AUPRC results, *Bayesian-CNN* outperforms the DeepEnhancer 4conv2pool4norm model 199 times out of 200. For AUROC



(a) AUPRC (b) AUROC

Figure 4.5: Comparison between *Bayesian-CNN* and DeepEnhancer. The plotted AUPRC (Left) and AUROC (Right) are averaged across multiple hold-out. Black bars represent the standard deviation.

values, 4conv2pool4norm outperforms *Bayesian-CNN* in only one task (54 out of 200). These results suggest that Bayesian optimization is able to select models competitive with state-of-the-art CREs classifiers. Moreover, we confirm that a CNN model trained on sequence data may achieve accurate results in the prediction of cis-regulatory element activity.

4.4.4 Test set balancing may lead to over-optimistic results

Yifeng Li et. al [140] deal with the data imbalance that characterizes the prediction of active regulatory regions by balancing both training and test data. We reproduced this experimental setup that we name "full-balanced". In addition, we test a "balanced" setup where we only balance the training set, and we compare both with the "unbalanced" experimental setup, for which results have been presented so far.

To summarize the different levels of imbalance between the different experimental setups, across all the tasks and cell lines, Table 4.5 reports the imbalance factor, which is measured as the ratio of cardinality of the higher represented class divided by the cardinality of the lower represented class, when the unbalanced and full-balanced setup are used.

Table 4.6 reports the average AUPRC across Bayesian FFNN and CCN models for the three different balancing techniques. In the last row, average AUPRC values over all tasks are reported for each balancing setup.

Comparing the AUPRCs in table 4.6 and in figure 4.6, we note that the balanced experimental setup is the one with the worst performance in all tasks. Wilcoxon tests confirm that, on average, the full-balanced setup produces the highest AUPRC scores.

We assume that a reduced performance of the balanced setup may be

Table 4.6: Average AUPRC compared across the different experimental setups. For each experimental setup (columns) and task (rows), we report the AUPRC averaged with respect to the Bayesian optimized FFNN and CCN models and the four cell lines. Bold text highlights significantly best results according to Wilcoxon rank sum tests at 0.01 significance level between full-balanced and unbalanced experimental settings; both unbalanced and full-balanced significantly outperform the balanced setting.

Task	balanced	full-balanced	unbalanced
IE vs IP	0.627	0.787	0.791
AP vs IP	0.745	0.884	0.901
AE vs IE	0.660	**0.885**	0.814
AE vs AP	0.834	**0.945**	0.856
AE + AP vs else	0.671	**0.882**	0.824
All tasks	0.707	**0.877**	0.837

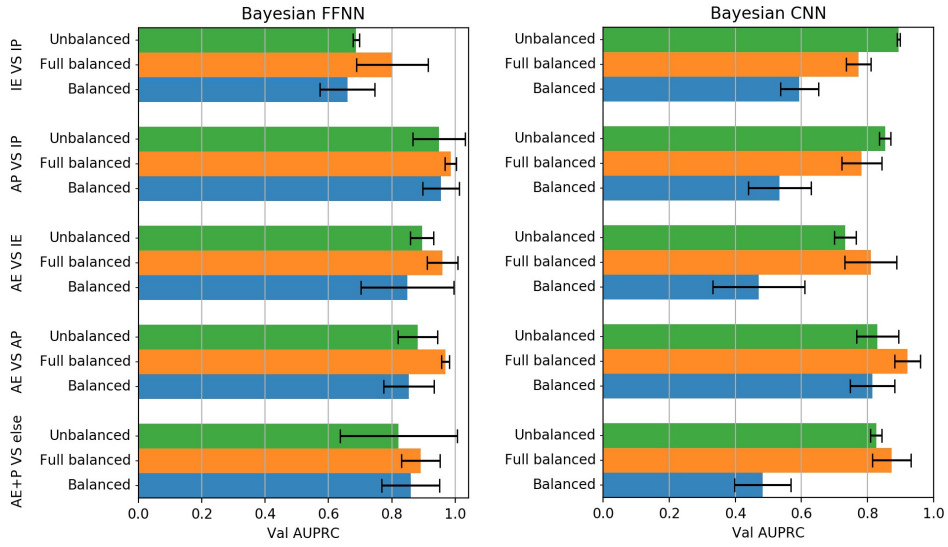


Figure 4.6: Comparison of the mean AUPRC obtained by *Bayesian-FFNN* (left) and *Bayesian-CNN* (right) among the three different balancing setups for each the five classification tasks. Black bars represent the standard deviation.

due to the fact that training set balancing is performed by sub-sampling, which requires to discard a relatively large amount of training samples. This ultimately affects data coverage during training and the neural network may not be able to effectively learn the intra-class variability, which results in a reduced generalization capability.

In contrast, the better performance obtained by the full-balanced experimental setup (Figure 4.6) could be the result of a distortion in the distribution of the test data (i.e. artificial increment of the ratio minority/majority class), thus leading to an over-optimistic estimation of the generalization capabilities of the predictor. Furthermore, test set balancing is not always feasible or possible, for instance when predicting the activity of not previously annotated CREs, since in this application the true labeling is not known.

4.4.5 CPU/GPU parallelization dramatically reduces computational time

All the tests performed so far have been done with the parallel optimization introduced in the previous section; thanks to that, model selection is performed in parallel using a multi-process master-slave paradigm to test multiple models concurrently. To measure the efficiency of this strategy, we performed a model selection over a grid search composed by 196 hyper-parameters candidates, comparing the execution times of a plain sequential model selection - where each candidate is evaluate one after another - and the optimized parallel strategy. We used the same dataset of the previous experiments.

More in detail we designed an experimental setup in which the workload between parallel and sequential where approximately the same: the grid is composed 93 different network configurations in terms of number of hidden layers and number of neurons per hidden layer, and two values for learning rate, thus the total number of point to be evaluated is 196. Table 4.7 summarize the grid space among the best model would be selected. We minimized the intervention of any "early stopping" criteria by increasing the "patience" parameter to 50 and the "delta" parameter to 10^{-5}, so that the condition for early stopping is hardly met: this ultimately provide a workload that did not vary between runs and between models, as the model training of each hyper-parameter combination was performed up to the maximum number of epochs with high probability. For the same reason, we did not run this test with the Bayesian optimization, as the model selection time strongly depends on the total number of evaluation that the optimizer proposes, and this greatly varies among runs, thus influencing the evaluation of the actual

speed-up between sequential and parallel runs.

Table 4.7: The architecture and learning parameters of the model used for measuring the efficiency of the parallel FFNN model selection method. The hyper-parameter space explored by the grid search is shown in curly brackets.

Bayesian-FFNN		
Layers	**Grid Space**	**Activation**
No. of fully connected layers	{0, 1, 2, 3}	
No. of units layer 1	{256, 128, 64, 32, 16, 8, 4, 2}	ReLU
No. of units layer 2	{128, 64, 32, 16, 8, 4, 2}	ReLU
No. of units layer 3	{64, 32, 16, 8, 4, 2}	ReLU
Output	1	Sigmoid

Learning parameters	
learning rate	{0.1, 0.2}
learning rate decay	0.01
l2 regularizer	0.001
batch size	100
weight value estimation	SGD
no. epochs	1000

We run 10 repetitions of each of the five tasks for the GM12878 cell line in the rebalanced modality. Tests have been performed on the Cineca Galileo partition: each computing node features 2x 18-cores Intel Xeon E5-2697 v4 (Broadwell) at 2.30 GHz and 2x NVidia K80 GPUs. The sequential run uses one master process and one slave process, exploiting the acceleration of one K80 GPU. The parallel run uses one master process and 12 slave processes and 2 GPUS; the pool of worker processes was splitted in two groups of 6 processes, and each group was assigned to a GPU, hence equally distributing the workload on the available accelerators. Each execution runs entirely intra-node. For each run we measured the computational time, including the training and test of the final best model. Results are summarized in Figure 4.7: we notice that time required for performing the grid search is greatly reduced in the parallel version. Speed-up varies between 7.5x and 9x.

4.4.6 Conclusions

In this work we presented two Neural Network models for the detection of the activity of Cis-Regulatory Elements, and our experimental results confirm

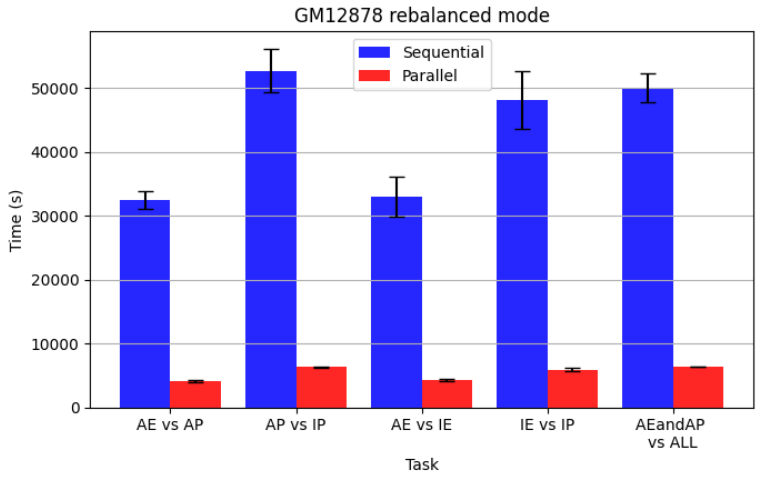


Figure 4.7: Comparison of the execution times required for performing the model selection through a grid search defined on Table 4.7 using the sequential (blue bars) and parallel (red bars) versions of the optimizer.

that deep neural networks can accurately predict active regulatory regions in specific cell lines.

We investigated the effect of model selection on the achieved performance, using BO techniques to automatically select the best network architecture and learning parameters for both FFNN and CNN models. Our comparative analysis confirmed that the learning models tuned by BO systematically outperformed their non-optimized counterparts. This shows that the Bayesian exploration of the hyper-parameters space is effective and produces optimal models. State-of-the-art methods usually employ "fixed structures" of the networks [136, 135], chosen based on previous work, expert domain knowledge, or based on a systematic exploration of the hyper-parameter space by means of grid search [140, 133]. To the best of our knowledge, this is the first time that BO is used to tune deep learning models to predict CREs.

The Bayesian FFNNs outperform Bayesian CNNs in tasks where we need to predict active versus inactive regulatory regions (i.e A-E vs I-E or A-P vs I-E). This is not surprising as epigenetic features are more informative than pure sequence data in predicting whether a regulatory region is active or not in a specific cell-type. On the contrary, when we need to discriminate between different types of regulatory regions (i.e. I-E vs I-P or A-E vs A-P) Bayesian CNNs outperform Bayesian FFNNs, as CNNs seem to extract DNA sequence motifs or characteristics (like GC or CpG content) that distinguish enhancers from promoters. These results indicate that FFNN and CNN models capture

different characteristics of CREs and suggest that a multi-modal approach, i.e. a model that is able to evaluate heterogeneous data set, may be able to outperform the current state-of-the-art in the detection of active CREs.

We also showed that model selection can be efficiently performed in parallel even on a single machine, and this ultimately has dramatic effects, leading to a substantial decrease of computational time.

Further, our results show that balancing the test set may lead to an over-optimistic estimation of the generalization performance of the model, and naive balancing of the training data may lead to poor generalization results.

Finally, we note that we applied Bayesian optimization only to a relatively small subset of the learning hyperparameters of the deep neural models. We hypothesize that enlarging the spectrum of the optimized learning parameters, as well as the range of exploration, we may further improve prediction performance in the challenging task of the prediction of tissue-specific regulatory regions.

Chapter 5

Scalable semi-supervised learning in biological networks through efficient parallel GPU computing

Several problems in network biology and medicine can be cast into a framework where entities are represented through partially labeled networks, and the aim is inferring the labels (usually binaries) of the unlabeled part. Connections represent functional or genetic similarity between entities, while the labellings often are highly unbalanced, that is one class is largely under-represented: for instance in the automated protein function prediction (AFP) for most Gene Ontology terms only few proteins are annotated, or in the disease-gene prioritization problem only few genes are actually known to be involved in the etiology of a given disease. Imbalance-aware approaches to accurately predict node labels in biological networks are thereby required. Furthermore, such methods must be scalable, since input data can be large-sized as, for instance, in the context of multi-species protein networks.

We propose a novel semi-supervised parallel enhancement of COSNet [300], an imbalance-aware algorithm built on the Hopfield neural model recently suggested to solve the AFP problem. By adopting a suitable representation of the graph and assuming a sparse network topology, we empirically show that it can be efficiently applied to networks with millions of nodes. The key strategy to speed up the computation is to partition nodes into independent sets so as to process each set in parallel by exploiting the power of GPU accelerators. This parallel technique ensures the convergence to asymptotically stable attractors, while preserving the asynchronous dynamics of the original model. Detailed experiments on real data and artificial big instances of the problem highlight scalability and efficiency of the proposed method.

By parallelizing COSNet we achieved on average a speed-up of 180x in

solving the AFP problem in the *S. cerevisiae*, *Mus musculus* and *Homo sapiens* organisms, while lowering memory requirements. In addition, to show the potential applicability of the method to huge biomolecular networks, we predicted node labels in artificially generated sparse networks involving hundreds of thousands to millions of nodes.

We outline that the proposed parallel algorithm based on parametric Hopfield Networks can be applied to several relevant problems in genomic medicine, ranging from disease gene prioritization to pharamcogenomics and drug repurposing.

As an extension to this research project and a possible future work, we also present a novel MCMC-based strategy for partitioning a graph into its maximal indipendent sets. This strategy is particularly tailored for CosNET-GPU, as the algorithm is designed to provide balanced solutions - i.e. maximal indipendent sets of almost the same number of nodes processed in parallel - thus allowing a better utilization of hardware resources and, at the same time, overcoming the inefficiency caused by small node clusters.

These research projects have been published or presented on [301], [302] and [303].

5.1 Par-COSNet

5.1.1 Background

The Automated Function Prediction of proteins (AFP) conveys the need of annotating the huge amount of protein sequences with their biomolecular functions. High-throughput sequencing technologies are rapidly increasing the gap between known protein sequences and proteins with experimentally annotated functions; indeed, more than 60 millions of protein sequences are available at the UniProt repository [304], and for instance less than 1% of these sequences have manually curated annotations in SwissProt [305]. Accordingly, the computational assignment of the biological functions to the proteins of an organism can greatly help in reducing this gap [306]. From this point of view, AFP can be modeled as a set of binary classification problems on graphs (one for every function), where nodes represent proteins, egdes encode their functional similarity, and nodes are labeled according to the current function (see fig 5.1). Two main characteristics of AFP are the large imbalance between positive (proteins annotated with the function under study) and negative nodes (the remaining proteins), and the large dimension of the input graph, since networks can contain millions of proteins (see for instance multi-species protein networks [307]).

Numerous graph/network-based approaches have been proposed by the scientific community to deal with the AFP problem, ranging from approaches relying on the guilt-by-association principle [308], assuming proteins topologically close in the graph are likely to share their functions, to label propagation algorithms based on Markov [309] and Gaussian Random Fields [310, 311, 312, 313]. Other studies based their evaluation on global [314] and local convergence properties, the latter one exploiting Hopfield networks (HNs) [315], and parametric variants of this model [316, 317, 318], including an extension of the Hopfield model to a multi-category context [319, 320], where nodes are inherently partitionable into separated categories (e.g. in the multi-species protein networks). In addition, since most AFP approaches exploit a protein neighborhood to infer the functions of that protein, some works introduced a generalization of the notion of pairwise-similarity among nodes by taking into account the contribution of neighbors shared by nodes[321, 322]. Finally, other relevant studies adopted techniques based on random walks [323, 324, 325], kernel matrices [326, 327], communities [328] and co-citations [329].

Despite their large diversity and their effectiveness in solving the AFP problem, most of the above mentioned methods neglect the class-imbalance problem, leading to classifiers tending to learn mainly the negative class, thus often obtaining a sensible deterioration of their performance [330]. Moreover, they do not suitably scale with the input size, in terms of both memory usage and execution time, since they usually adopt matrix representations to embed the graph (without exploiting the graph sparsity), and basically utilize sequential programming in their model design. Indeed, recent and interesting works proposed to use secondary memory-based technologies to exploit the large disks available in standard computers to apply to big data standard graph-based semi-supervised learning algorithms; nevertheless, this can be done at the expense of the efficiency, since the swapping between secondary and primary memories data increases the computational burden [331].

In this study we propose Par-COSNet (Parallel COSNet), a methodology for solving AFP problem specifically designed to cope with the label imbalance problem and the big size of input data. It extends COSNet (Cost-Sensitive Neural Network) [317], a state-of-the-art semi-supervised method for AFP based on HNs. COSNet introduces a parametric HN to effectively handle the label imbalance, but its available implementation [300] still adopts a matrix representation of input data, allowing its application (on ordinary off-the-shelf computer) only to networks with few tens of thousands of nodes.

As first contribution, Par-COSNet reduces the memory requirements of COSNet by adopting a sparse representation of both network connections

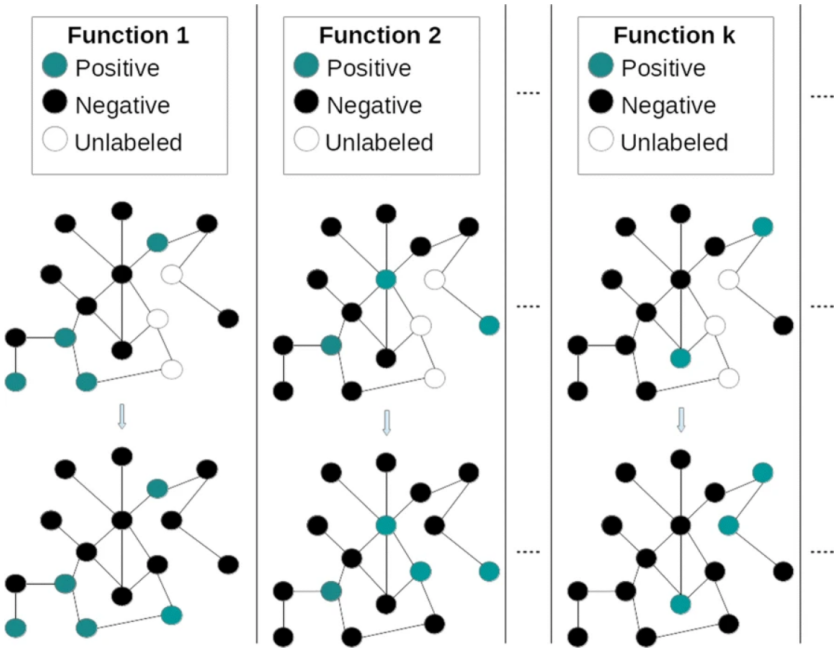


Figure 5.1: AFP problem as a set of binary classification problems. The aim is determining the color/label of unlabeled nodes/proteins, given the graph topology and the labels of the known part of the graph.

and node labeling, leveraging the sparsity of input graphs and the scarcity of positive proteins characterizing data in the AFP context. On the other side, the overall excution time is remarkably reduced 1) by splitting the HN dynamics over independent sets of neurons, where nodes in the same independent set are updated in parallel, and 2) by exploiting Graphics Processing Unit (GPUs) devices under the CUDA (Compute Unified Device Architecture) parallel programming model [332] to use one or multiple GPUs in parallel along with the CPU. Specifically, multiple GPU cores are assigned to a single independent set, thus updating in parallel the neurons within each independent set, while independent sets are in turn sequentially updated to preserve the HN convergence properties. Considering that usually multiple GO terms should be predicted for each protein, the proposed implementation adds another level of parallelism through the multithreading execution, where each CPU thread is given an instance of the AFP problem (a function to be predicted), and multiple threads are run in parallel (each using multiple cores in parallel). This thereby results in a noticeable speedup with regard to the original COSNet implementation.

We evaluated the Par-COSNet gain in terms of both memory requirements reduction and execution speedup by testing COSNet and Par-

COSNet in predicting the Gene Ontology functional terms [333] for three eukaryotic organisms. Furthermore, synthetic graphs of different sizes, from hundreds of thousands to millions of nodes, and of different densities have been generated to empirically show the applicability of Par-COSNet on large networks.

Par-COSNet source code has been publicly released for evaluation and testing purposes, and is available on the official AnacletoLab GitHub repository, at [334].

5.1.2 Methods

In this section we recap the original formulation of COSNet , introduce and in-depth describe the parallel version and its GPU implementation - Par-COSNet - with all the associated technique for providing a fast and scalable approach to the problem.

Semi-supervised learning on graphs

Semi-supervised learning on graphs comprises a class of Machine Learning algorithms that uses the features observed on a sub-set of nodes in a graph for estimating missing values in other nodes [335].

Formally, we consider the set of nodes $V = \{1, 2, \ldots, n\}$ in a graph $\mathcal{G} = \langle V, E_W \rangle$, with edge set E_W induced by W, a symmetric $n \times n$ real weight matrix, whose elements w_{ij} encodes the relationship (for instance, the similarity) between pairs $(i, j) \in V \times V$.

The nodes V are labeled with $\{+, -\}$, leading to the subsets P and N of positive and negative nodes. However, it may happen that this labeling is known only for a subset $S \subset V$, while is unknown for $U = V \setminus S$. Moreover, let be $S^+ = S \cap P$ and $S^- = S \cap N$. In this case, the labeling problem consists in finding a bipartition (U^+, U^-) in U, where U^+ and U^- are the subsets of unlabeled nodes considered candidates for the classes $U \cap P$ and $U \cap N$, respectively.

It is worth noting that this kind of labeling problem is highly relevant in Precision and Genomic Medicine, as several problems can be recasted as such, for instance, gene-disease prioritization or drug repositioning.

The *Automated protein function prediction problem* can be also recasted as such by modeling a set of proteins as the set V of the nodes of a graph, w_{ij} as the functional similarities between pairs (i, j) of proteins, and functional classes the different -partial - labeling of the graph. In this context, AFP is set as a semi-supervised learning problem on graphs, since protein functions

can be predicted by exploiting both labeled and unlabeled nodes/proteins and the weighted connections between them [336].

COSNet

COSNet (COst Sensitive neural Network) [316, 317] is a neural algorithm recently proposed to face with the problem of labeling partially labeled graphs, like the AFP problem. More specifically, this technique relies on a parametric family of the Hopfield model [337], where the network parameters are learned to cope with the label imbalance and the network equilibrium point is interpreted to classify the unlabeled nodes.

Formally, for a given a set of nodes $V = \{1, \ldots, n\}$, COSNet is a triple $\mathcal{H} = \langle W, \boldsymbol{\lambda}, \rho \rangle$, where:

- $W \in \mathbb{R}^{n \times n}$ is a symmetric weight matrix whose elements $w_{ij} \in [0, 1]$ represent the connection strength between the neurons (nodes) i and j (naturally $w_{ii} = 0$),
- $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_n\} \in \mathbb{R}^n$ denotes the neuron activation thresholds,
- $\rho \in [0, \frac{\pi}{2})$ is a parameter which determines the two neuron activation (state) values $\{\sin \rho, -\cos \rho\}$.

The rationale of the parameter ρ is to conceptually separates node labels and neuron activation values, since for classical HNs activation values are in the set $\{-1(0), 1\}$, that means node labels and neuron activation values coincide. Thus, appropriately learning the parameter ρ allows the algorithm to counterbalance the large imbalance towards negatives (see [317]).

A relevant issue for the correct design of this kind of recurrent neural networks is the synchronization of its computing nodes. The Hopfield model is a discrete-time dynamical system which admits synchronous or asynchronous updating or even both if an hybrid setting is admitted. In case of asynchronous (sequential) updating, each unit is updated independently from the others at any time t. Thus, by denoting with $\pi = \pi(1), \cdots, \pi(n)$ an arbitrary permutation on nodes V and with $x_{\pi(i)}(t)$ the state of neuron $\pi(i)$ at time t, the dynamics assumes the form:

$$x_{\pi(i)}(t+1) = \begin{cases} \sin \rho, & \text{if } h_{\pi(i)}(t+1) \geq 0 \\ -\cos \rho, & \text{otherwise} \end{cases}, \tag{5.1}$$

where

$$h_{\pi(i)}(t+1) = \sum_{j=\pi(1)}^{\pi(i-1)} w_{\pi(i)j} x_j(t+1) + \sum_{j=\pi(i+1)}^{\pi(n)} w_{\pi(i)j} x_j(t) - \lambda_{\pi(i)}.$$

Convergence depends on the weight matrix structure W and the rule by which the nodes are updated. In particular, if the matrix is symmetric, it has been proved that the network converges to a stable state when operating in asynchronous mode, while it converges to a cycle of length at most 2 when operating in a synchronous (fully-parallel) mode. The proof of these properties is grounded on the so-called energy function, which is non decreasing when the state of the network $\boldsymbol{x} = (x_1, x_2, \cdots x_n)$ changes as a result of the computation (5.1). Since the energy function is upper-bounded, it follows that the system will converge to some state. In the classic discrete Hopfield model the energy function has the following quadratic form:

$$E(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^T W \boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{\lambda}. \tag{5.2}$$

As a major result, it has been shown that (5.2) is a Lyapunov function for the Hopfield dynamical systems with asynchronous dynamics, i.e., for each $t > 0$, $E(\boldsymbol{x}(t+1)) \leq E(\boldsymbol{x}(t))$ and exists a time $\bar{t}$ such that $E(\boldsymbol{x}(t)) = E(\boldsymbol{x}(\bar{t}))$, for all $t \geq \bar{t}$. Moreover, the reached fixed point $\hat{\boldsymbol{x}} = \boldsymbol{x}(\bar{t})$ is a local minimum of (5.2).

The overall scheme of the CosNet algorithm adopting dynamics (5.1) can be sketched as follows:

INPUT. A symmetric weight matrix $W \in [0,1]^{n \times n}$, a labeling function $y : V \to \{+, -\}$; the subsets S^+, S^- and U of positive, negative and unlabeled instances, respectively; an initial value $\boldsymbol{x}(0) \in \{\sin\rho, -\cos\rho, 0\}^n$; a permutation π on the set U.

Step 1. Learn the parameters $\rho = \bar{\rho}$ and $\lambda_i = \bar{\lambda} \in \mathbb{R}$ on the sub-network restricted to labeled nodes S such that the state represented by known labels is "as close as possible" to a minimum of the network energy.

Step 2. Regularize the network dynamics in order to prevent the network stucking into trivial energy minima by suitably changing the thresholds and the connection weights (see [317] for more details). Hereafter, abusing notation, we assume this regularization is embedded in the connections w_{ij} and thresholds λ_i, for each $i, j \in V$.

Step 3. Run the sub-network restricted to unlabeled nodes embedding the learned parameters $\bar{\rho}$ and $\bar{\lambda}$ until an equilibrium state $\hat{\boldsymbol{u}}$ is reached. Based on state $\hat{\boldsymbol{u}}$ compute the bipartition (U^+, U^-) of U into the positive and negative neurons.

OUTPUT. A bipartition (U^+, U^-).

Step 1 and 2 allow the method to deal with labeling imbalance, since the first step counterbalances the predominance of negatives in the node neighborhood, whereas the second step ensures to avoid trivial states composed of all negative predictions.

Parallel COSNet

The goal of this work is to speed-up the COSNet algorithm by introducing a partial synchronous updating of the computational units in order to parallelize the network evolution stage while preserving the asynchronous dynamics. Since in context like AFP the weight matrix W is usually sparse, several nodes are independent from each others.

The basic idea is to partition the nodes belonging to the undirected graph $\mathcal{G} = \langle V, E_W \rangle$ (with edge set E_W induced by W) into a small number of independent subsets, which can be done recasting the problem as a vertex coloring problem of $\mathcal{G}$.

Let a vertex coloring be a map $c : V \to C$, where C is a set of distinct colors. We say that a coloring is proper if adjacent vertices of $\mathcal{G}$ receive distinct colors of C, which in turn means that if $(i, j) \in E_W$, then $c(i) \neq c(j)$. Clearly, in any proper vertex coloring of $\mathcal{G}$ the vertices that receive the same color are independent. A k-coloring of a graph $\mathcal{G}$ is a vertex coloring of $\mathcal{G}$ that uses at most k colors and $\mathcal{G}$ is said to be k-colorable if it admits a proper vertex coloring using at most k colors. Hereafter, we denote a proper k-coloring by a partition $\mathcal{P} = \{V_1, \ldots, V_k\}$ of the vertex set V into k independent subsets.

Under this setting it is easy to show that by simultaneously updating the nodes of each element V_i of $\mathcal{P}$, one at a time, whatever the permutation of $\{1, \ldots, k\}$ is, the Hopfield network asynchronous dynamics is preserved. This implies that, given an initial state, the network is guaranteed to converge to a unique fixed point which is a local minimum of (5.2). To see that this property holds, we can start by observing that each partition $\mathcal{P}$ induces a permutation $\pi^{\mathcal{P}}$ (that for sake of notation we simply denote by π) on the set V such that, for all $i = 1, \ldots, k$, the subsequence $\pi_i = \pi_i(1), \ldots, \pi_i(n_i)$ collects the nodes within $V_i = \{\pi_i(1), \ldots, \pi_i(n_i)\}$, being $n_i = |V_i|$. The permutation

π, for a fixed $\mathcal{P}$, is then written as the juxtaposition of all subsequences $\pi = \pi_1, \cdots, \pi_k$.

It is easy to see now that, synchronously updating all nodes in the subsequence π_i, but asynchronously accordingly to an arbitrary permutation $\omega = \omega(1), \ldots, \omega(k)$ on all subsequence indexes $\{1, \ldots, k\}$, the constraints prescribed by the asynchronous updating rule are respected. Indeed, given the permutation $\pi(\omega) = \pi_{\omega(1)}, \cdots, \pi_{\omega(k)}$ induced by a k-coloring $\mathcal{P}$ and the permutation ω of the partition indexes, by applying the following partially-parallel update

$$\forall u \in \pi_{\omega(i)}, \quad x_u(t+1) = \begin{cases} \sin \rho, & \text{if } h_u(t+1) \geq 0 \\ -\cos \rho, & \text{otherwise} \end{cases}, \tag{5.3}$$

where

$$\begin{aligned} h_u(t+1) = & \sum_{v \in \pi_{\omega(1)} \cdots \pi_{\omega(i-1)}} w_{uv} x_v(t+1) \\ & + \sum_{v \in \pi_{\omega(i+1)} \cdots \pi_{\omega(k)}} w_{uv} x_v(t) - \lambda_u. \end{aligned} \tag{5.4}$$

done in parallel on $u \in \pi_{\omega(i)}$ and sequentially over $\pi_{\omega(1)}, \cdots, \pi_{\omega(k)}$, we obtain the same attractor as in (5.1) with sequential update dictated exactly by $\pi = \pi(\omega)$.

Therefore, the Par-CosNet algorithm - which stands for "Parallel CosNet" - encompassing a graph coloring strategy and adopting the Hopfield dynamics (5.3), can be sketched as follows:

INPUT. Idem as in CosNet; a k-coloring $\mathcal{P} = \{U_1, \ldots, U_k\}$ of U inducing the permutation π; a permutation ω on the first k integers.

Step 1. Idem as in CosNet.

Step 2. Idem as in CosNet.

Step 3. Run the sub-network restricted to unlabeled nodes embedding the learned parameters $\bar{\rho}$, $\bar{\lambda}$ and then updating (5.3) in parallel for nodes U_i by following the update permutation $\pi(\omega) = \pi_{\omega(1)}, \cdots, \pi_{\omega(k)}$ until an equilibrium state $\hat{\boldsymbol{u}}$ is reached. On the basis of state $\hat{\boldsymbol{u}}$ compute the bipartition (U^+, U^-) of U into the positive and negative neurons.

OUTPUT. A bipartition (U^+, U^-).

Despite the additional computational cost for finding a suitable k-coloring, Par-COSNet shows a significant performance speed-up compared to original algorithm, mainly due to the structure of partition $\mathcal{P}$ and the number of processors $\mathcal{N}$ at hand. Although in many applications it is required to find a partition $\mathcal{P}$ having minimum cardinality (also called the chromatic number of $\mathcal{G}$), our parallelization task is instead aimed at making the evolution of the Hopfield neural system in COSNet as fast as possible; therefore, we rather seek a k-coloring able optimize the computational efficiency of the processors. In the following we discuss some implementation issues regarding a specific parallel architecture provided by GPUs (Graphics Processing Units) rather than carrying out cost analysis for abstract theoretical models.

GPU implementation

Originally designed for graphics applications, GPUs have gradually acquired increasing importance for scientific computing and computer simulations and their processing power is one order of magnitude higher than current generation CPUs. This considerable computational power, due to specific hardware design, has paved the way for big data applications. For instance, in many domains it is frequent to address problems on very large graphs, often involving millions of vertices and graphics accelerator hardware has become a cost-effective parallel platform to solve them [338].

For this reasons, an implementation of Par-COSNet, specifically targeted to NVIDIA GPUs [334] has been developed for assessing the effectiveness and performances of the algorithm. This implementation has been developed under the CUDA (Compute Unified Device Architecture) programming model [332]. Also under the same programming paradigm, a parallel implementation of a Greedy Graph Coloring (GGC) algorithm has been developed for solving the graph coloring problem, as required by the Par-COSNet algorithm.

A GPU processor is composed by an array of Streaming Multiprocessors (SM), each one having a variable number of CUDA cores, depending on the generation of the GPU. Therefore, a single GPU processor can be seen as an array of thousands of simplified processing cores capable of scheduling and concurrently running a very large number of threads. Threads are executed according to the SIMT (Single Instruction Multiple Thread) architecture - a variation of the SIMD execution model (Single Instruction, Multiple Data) - in which every thread in a block is executed independently and concurrently.

In Par-COSNet and in the GGC algorithms this execution model has been exploited to achieve a high level of fine grained parallelism: in the GGC algorithm, each node of the graph is assigned to a thread, while in

the Hopfield dynamics, an entire block of threads, ranging from 32 to 512, is assigned to a neuron for updating its state. Launching a very large number of threads has a useful side effect in the CUDA programming model, since it helps to hide latencies between the processing core and the on-board video RAM, where data are stored.

One of the major difficulty in GPU programming lies in managing its complicate memory model, since data can be stored in different address spaces, each one having its own trade-offs in terms of accessing speed and size; on top of that, CUDA kernels - i.e. programs that run on the GPU - are able to access only data residing on the on-board RAM (limited in size, compared to the host memory), therefore the host system must copy the relevant part of data on the GPU before the actual computation starts, and copy back the results after the computation ends. Repeatedly copying data back and forth the GPU causes latencies that slows down the computation.

In Par-COSNet a good trade-off has been achieved through several strategies. First of all, to cope with the limited amount of video ram, the graph is stored in the host RAM by means of a compressed representation, such that the entire net takes roughly $(2 \times n + 2 \times m)$ doubles, being n and m respectively the number of nodes and the number of edges of the net; this is possible by exploiting the sparsity of the net. Then, only the unlabeled portion of the graph is copied to the GPU and processed, therefore further reducing the transmitted data volume.

Another strategy to minimize latencies occurring in the GPU programming model is to reduce the number of synchronization points between the host system and the GPU. In an ideal heterogeneous system, the host and the hardware accelerator should work independently for maximizing concurrency, but this can be not feasible if the algorithm requires several synchronization points; when such points are hit during the execution, one device may be put to halt, waiting for the other to reach the synchronization point, thus slowing down the computation. This was the case of the Greedy Coloring, where each iteration on the GPU required a validation on the host system. We solved this issue by writing a coloring kernel which is rather independent from the system host, exploiting CUDA Dynamic Parallelism, where each kernel can spawn a set of sub-kernels (this is opposed to the traditional kernel launching model, where only the host system is allowed to launch computational tasks on the GPU). In this way, the entire coloring task does not require any synchronization by the host, which is therefore free to accomplish other tasks and concurrency between GPU and host is maximized.

Parallel Greedy Graph Coloring

As highlighted in previous section, graph coloring is a key strategy to make an efficient use of parallel (SIMT) architecture because it allows to split complex tasks into small independent subtasks that can be carried out concurrently. In our setting, each subtask can be identified by the so called Maximal Independent Set (MIS) of a graph, i.e., a maximal collection of vertices $I \in V$ subject to the restriction that no pair of vertices in I are adjacent. This subgraph structure is strictly connected to coloring, because it represents a common parallel approach for graph coloring, leveraging the parallel MIS algorithm as a subroutine (see the schema in Algorithm 4). In this approach, a partition $\mathcal{P} = \{V_1, \ldots, V_k\}$ is conceived as a collection of MISs yielded by subsequent call of the Luby parallel algorithm [339] (called LubyMIS in the pseudocode) which works in a greedy modality. The idea is that in every round i it finds a subset $\bar{I} \subset V$ which is an independent set. Then it adds $\bar{I}$ to the current independent set I, and removes $\bar{I}$ and its neighbors $\mathcal{N}(\bar{I})$ from the current graph V' (see pseudocode). If $\bar{I} \cup \mathcal{N}(\bar{I})$ is a constant fraction of $|V'|$, then it will only needs $\mathcal{O}(\log |V|)$ rounds to determine I. It will instead ensure that by removing such subset from the graph, it removes a constant fraction of the edges. The final subset I at round i is then considered a set V_i in the partition $\mathcal{P}$.

To choose $\bar{I}$ in parallel, each vertex v independently adds itself to $\bar{I}$ with a well chosen probability $p(v)$. Since we want to avoid adding adjacent vertices to $\bar{I}$, we will prefer to add low degree vertices. But, if for some edge (u, v), both endpoints were added to $\bar{I}$, then we keep the higher degree vertex. The above strategy is concisely summed up in the statement called **choice** in the GPULubyMIS pseudocode.

As for the implementation, the latter procedure has been implemented in SIMT form through the curand library available within CUDA toolkit. Each thread handles a node of the whole graph, draws a random number following a binomial distribution with fixed probability $p(v)$ and establishes whether it belongs or not to the MIS under construction.

Par-COSNet

Here we present and discuss the implementation of the parallel Hopfield algorithm using CUDA programming model, sketched in the pseudocode of Algorithm 5.

To face with the parallelization of the asynchronous dynamics (broadly described in previous sections), it should be noted that at the base of this neural architecture there is an intrinsic parallelism in the computation of the

Algorithm 4 Parallel greedy coloring

Input: graph $\mathcal{G} = \langle V, E \rangle$
Output: coloring (partition) $\mathcal{P} = \{V_1, \ldots, V_k\}$

$\mathcal{P} \leftarrow \emptyset$
$i \leftarrow 1$ ▷ first color
while $\mathcal{G} \neq \emptyset$ **do**
  $V_i \leftarrow \text{GPULubyMIS}(\mathcal{G})$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{V_i\}$
  $Z \leftarrow V_i \cup \mathcal{N}(V_i)$ ▷ union of V_i with its neighbors
  $\mathcal{G} \leftarrow \textbf{sub}\langle V - Z, E \rangle$ ▷ induced subgraph excluding Z
  $i \leftarrow i + 1$ ▷ set new color
end while

procedure GPULubyMIS($\mathcal{G}$)
  $I \leftarrow \emptyset$
  $\mathcal{G}' = \langle V', E' \rangle \leftarrow \mathcal{G} = \langle V, E \rangle$
  while $\mathcal{G}' \neq \emptyset$ **do**
    choice $\bar{I} \subseteq V'$ ▷ select an independent set of $\mathcal{G}'$
    $I \leftarrow \bar{I} \cup I$
    $Z \leftarrow \bar{I} \cup \mathcal{N}(\bar{I})$ ▷ union of $\bar{I}$ with its neighbors
    $\mathcal{G}' \leftarrow \textbf{sub}\langle V' - Z, E \rangle$ ▷ induced subgraph excluding Z
  end while
  return I ▷ MIS of $\mathcal{G}$
end procedure

activation function (5.4): each single neuron implements this simple thresholding function, whose state is either "active" or "not active". This state is determined by calculating the weighted sum of the states of its connected neurons. If the sum exceeds the threshold, the state will change to active, otherwise, the neuron will be non-active.

All these observations are captured in what we call GPUHopfieldNet procedure within the pseudocode. This algorithm naturally exposes two different levels of parallelism which can be exploited for realizing an effective scheme well suited for GPU execution. These levels correspond to two nested and different tasks: the first consists in concurrently compute the update of all nodes sharing the same color (i.e., belonging to i-th cluster V_i), while the second consists in running in parallel the addition and thresholding required for the update of the state of each neuron. Also, the tasks are characterized

Algorithm 5 Par-COSNet: Hopfield net (Step 3)

Input: net $\mathcal{H} = \langle W_U, \bar{\boldsymbol{\lambda}}, \rho \rangle$, k-coloring $\mathcal{P} = \{U_1, \ldots, U_k\}$ of vertices in U
Output: final state $\hat{\boldsymbol{u}}$

$\boldsymbol{u} \leftarrow 0$ ▷ net state
nonstop $\leftarrow$ *true* ▷ stop flag
while nonstop **do**
  $\boldsymbol{u'} \leftarrow \boldsymbol{u}$
  for $i \leftarrow 1, k$ **do**
    $\boldsymbol{u'} \leftarrow$ GPUHopfieldNet$(\mathcal{H}, U_i, \boldsymbol{u'})$
  end for
  if $\boldsymbol{u} \neq \boldsymbol{u'}$ **then**
    $\boldsymbol{u} \leftarrow \boldsymbol{u'}$
  else
    nonstop $\leftarrow$ *false*
  end if
end while

procedure GPUHopfieldNet$(\mathcal{H}, U, \boldsymbol{u})$
  for all $v \in U$ **do** ▷ in parallel
    use SIMT to compute h_v ▷ compute (5.4)
    $u_v \leftarrow \begin{cases} \sin \rho, & \text{if } h_v \geq 0 \\ -\cos \rho, & \text{if } h_v < 0 \end{cases}$ ▷ compute (5.3)
  end for
  $\hat{\boldsymbol{u}} \leftarrow \boldsymbol{u}$
  return $\hat{\boldsymbol{u}}$ ▷ partially update fixed point
end procedure

by different granularity, the latter being more fine-grained than the former.

The first and more coarse-grained task, i.e. concurrently updating all the nodes having the same color, has been tackled by sequentially launching k different instances of the GPUHopfieldNet kernel during each iteration, being k the number of colors returned by the coloring algorithm. Note that this approach respects the sequential requirements of the Hopfield dynamics. Each kernel is launched with a different configuration reflecting the number of nodes belonging to each cluster V_i. In particular, we assign $|V_i|$ CUDA thread blocks to the i-th kernel, i.e. one CUDA thread block per node.

To perform the second task, i.e. to update the state of each neuron, a fixed number n_T of threads (ranging from 32 to 1024) is in turn assigned

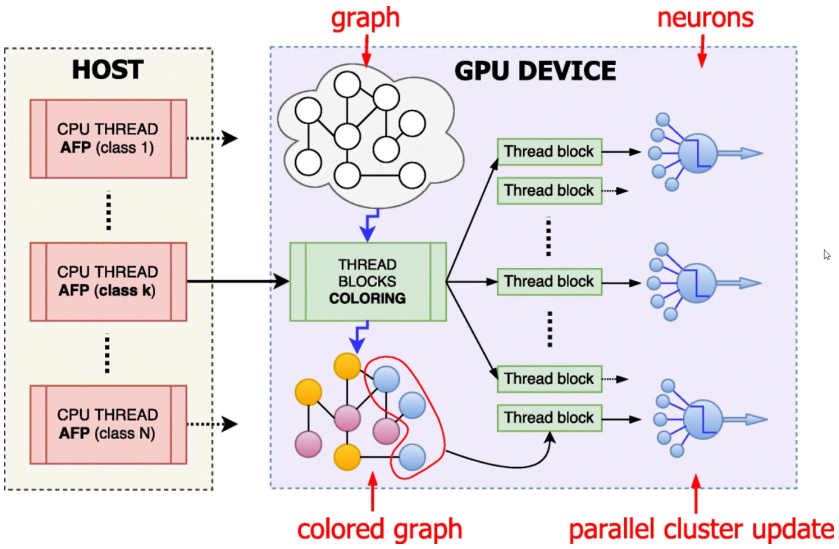


Figure 5.2: CPU/GPU schema of the Par-COSNet parallelization. Multiple CPU threads are launched in parallel each one solving the AFP problem for a given class/protein function. The GPU thread blocks, each composed of several CUDA threads, first solve the coloring problem for the graph and then concurrently process all neurons of a given color, for all colors in sequence. A further fine-grained level of parallelism is finally introduced by assigning to each neuron a thread block to perform the neuron level local computations.

to each thread block. To process the weigthed contributions of each neuron neightborhood, we proceed in SIMT modality using shared memory among all threads in the same block and applying the known primitive called *parallel reduction*: this is a tree-based approach used within each CUDA thread block frequently applied to process very large arrays of N elements. It can be shown that, having at disposal P CUDA threads physically in parallel (P processors), the time complexity of the parallel reduction is $\mathcal{O}(N/P+\log N)$. By varying n_T, the algorithm can be adapted to the density of the graph: very dense graphs will benefit from an increase of n_T, since more threads will speed-up the evaluation of the neighborhood and the computation of the parallel reduction.

CPU multithreading and data representation

On top of the parallelization of the Hopfield dynamics and graph coloring, Par-COSNet exploits the independence of protein functions (target classes) to further accelerate the computation, since as stated in the introduction

multiple AFP problems can be solved concurrently.

As shown in Fig. 5.2, in Par-COSNet we exploit this natural additional level of parallelism by means of CPU multi-threading, where each target class is assigned to a different CPU thread. In this computing modality, each thread reads a different class labeling, trains its associated net and then executes the coloring and Hopfield dynamics on the GPU. This dramatically improves the performances of the overall process since each AFP instance is independent, therefore multiple instances can be run concurrently. On the other side, CPU multi-threading implies to share the resources of a single GPU among multiple CPU threads, leading to serialization latencies. To deal with this problem, we exploited a CUDA compiler option that allows to assign at runtime each CPU thread to a different CUDA stream, where a CUDA stream is a queue of commands or operations that are executed in a specific order; while operations on a stream are executed sequentially, operations in different streams may be executed concurrently or out of order with respect to one another. In the specific case of Par-COSNet, kernel executions belonging to different AFP problems are interleaved and executed concurrently.

Also, the algorithm in principle could benefit of additional GPU devices to split the workload, for instance, in a system with 2 GPUs, all the odd-numbered CPU threads might offload the parallel computation to the first GPU, while the even-numbered to the second GPU.

An issue that arises when working with big datasets is memory consumption. We chose to adopt a compressed format for storing the net and the labeling y. Indeed, by exploiting the sparsity of W, for each node solely its neighbors are kept in memory at run time; moreover, leveraging the scarcity of positives, only the nodes belonging to the minority class are maintained, thus saving a huge amount of memory when the input graph contains millions of nodes.

5.1.3 Results and Discussion

In this section we present the experimental results obtained by deeply testing Par-COSNet; we provide speed-up memory allocation and efficiency indexes in both simulated and real contexts, using synthetic and real datasets.

All tests have been performed on a workstation having 2 Intel Xeon $E5-2620v3$ CPUs clocked at 2.40 GHz, 64 GB of RAM memory, 2 TB disk and Linux Ubuntu 16.04 as operating system. The workstation is equipped with an NVidia GeForce $GTX980$ GPU card, featuring 2048 CUDA cores, 4 GB of dedicated on-board video memory and having Compute Capability 5.2. The C++ portion of the code has been compiled wit GCC 5.4.0, while the

rest (that is, the CUDA kernels) with NVCC 8.0.44. As for COSNet, it has been executed under R version 3.4.2.

As for the evaluation of the computational speed-up of Par-COSNet with respect to the already available COSNet implementation, we followed the usual guidelines for the evaluation of computing performances as in [261]. Par-COSNet has been executed using 1, 4, 8 and 12 CPU thread.

To assess the scalability of the multithread approach, the overall CPU occupancy of each execution has been measured and reported. Also the maximum memory footprint and the execution time have been collected; the latter is used to evaluate the speed-up defined as $S = T_s/T_p$, where T_s and T_p are the execution times of sequential and parallel implementation, respectively. Computing times have been recorded using the high precision timer integrated in the C++14 STL library. Occupancy and maximum memory footprint have been measured through the htop Linux command line utility.

As a refence for both prediction quality and execution time we considered the COSNet implementation publicly available as an R package [300], which efficiently implements the time consuming procedures (e.g. parameter learning and Hopfield dynamics) in C language. However, this package adopts a matrix representation for the input network, since the R language is optimized for matrix-based computations.

Experimental data

This section is devoted to the description of both real and artificial networks used throughout the paper.

Real Data. Three organisms have been considered for the AFP problem, namely *Homo sapiens* (human) and two model organisms *S.cerevisiae* (yeast), *Mus musculus* (mouse). The input networks have been retrieved from the STRING database, version 10.0 [307]: the STRING networks are highly informative networks merging several sources of information about proteins, coming from databases collecting experimental data like BIND, DIP, GRID, HPRD, IntAct, MINT or from databases collecting curated data such as Biocarta, BioCyc, KEGG, Reactome. The total number of proteins is 6391, 21151 and 19576 for yeast, mouse and human organisms, respectively.

All networks have one large connected component, with human and mouse networks with one or more smaller connected components. Furthermore, the human network is the most compact, having the smallest ratio between the number of nodes and the network diameter (see Table 5.1 for the network topological characteristics).

In the STRING database each protein-protein connection is associated with a confidence score: in principle, discarding edges with confidence score

Table 5.1: Characteristics of protein networks. Column **Compon.** denotes the number of connected components in the network, whereas **Largest compon. size** is the number of nodes in the largest connected component. **Diameter** is the number of edges on the longest path between two nodes, without considering edge weights.

Organism	Nodes	Average degree	Compon.	Largest compon. size	Diameter	Weighted diameter
Yeast	6391	314.0563	1	6391	6	1.0925
Mouse	21151	596.3804	21	21105	9	1.8362
Human	19576	579.9477	2	19574	6	1.0302

lower than a fixed threshold would allow to select more reliable connections; on the other side, it would generate isolated proteins (proteins with no connections), which accordingly should be discarded from the analysis. Since the aim of this study is supplying a methodolgy able to work on large networks, no edge threshold has been applied, thus including all available proteins.

Protein networks have been normalized as follows: denoted by $\hat{W}$ the matrix obtained from the STRING connections, the final network W is obtained by applying the normalization

$$W = D^{-1/2}\hat{W}D^{-1/2},$$

where D is a diagonal matrix with non-null elements $d_{ii} = \sum_j \hat{W}_{ij}$. Note that W is still symmetric.

Protein functional annotations have been retrieved from the Gene Ontology (GO) database, using the UniProt GOA releases 69 (9 May 2017), 155 (6 June 2017) and 168 (9 May 2017) respectively for yeast, mouse and human organisms. The GO terms have been selected from all the three branches *Biological Process* (BP), *Molecular Function* (MF), and *Cellular Component* (CC), by considering terms with at least 50 annotated proteins with experimental evidence, in order to obtain a minimal amount of information for the prediction of GO terms. The number of the resulting GO terms is summarized in Table 5.2. The mapping from UniProt to STRING protein identifiers was carried out according to the mapping files provided at UniProt repository.

Artificial data. In order to assess the performances in terms of computational time and memory consumption over larger datasets, the parallel implementation has also been tested on several artificial datasets which have been randomly generated. For random graphs we use the Erdős model in which the graph size n and the probability p to have an edge between a pair

Table 5.2: Number of GO terms with at least 50 annotations.

Organism	CC	MF	BP
Yeast	50	74	191
Mouse	85	112	733
Human	115	193	580

of nodes are fixed. In particular, the range chosen for n goes from $5 \cdot 10^5$ to $1.5 \cdot 10^6$, while for p we chose values so as to reproduce a graph density $\sigma = np$ close to that of biological networks. Edge weights were uniformly generated in $[0, 1]$, while the corresponding set of labels has been generated so as to respect the unbalancing of realistic cases.

We run Par-CosNet with the resulting 9 artificial datasets and recorded the computation time and memory consumption.

Real data

Although CosNet has already been validated in [317, 318, 340] to solve the AFP problem, for the sake of completeness we report in Table 5.2 its capability in predicting the GO terms. The generalization abilities of CosNet have been assessed through a 5-fold cross validation (CV), and evaluated in terms of *Precision* (the proportion of positives correctly predicted) and *Recall* (the proportion of real positive discovered) combined in the F measure which is the harmonic mean of precision and recall. In this context, where positives are rare, these measures are more informative than the error rate. Moreover, to evaluate the ability of COSNet as ranker, we also report the *Area Under the Precision Recall Curve* (AUPRC), measure adopted in the recent CAFA2 international challenge to evaluate protein ranking [341]. To provide a protein ranking for COSNet related to the current GO term, we adopted the internal energy on neuron at equilibrium, as done in [342, 318]. Table 5.3 contains the corresponding results averaged across terms in each GO branch.

For validating the correctness of Par-CosNet, we run the same tests on the same dataset and using the same configuration, i.e. 5-fold cross validation and the same learning hyper-parameters, confirming that its classification performances are the same as CosNet.

Table 5.4 reports the average execution time in seconds of CosNet and Par-CosNet for computing an entire CV cycle for a single GO term.

To better evaluate the contribution of multithreading, Table 5.6 shows the average CPU occupancy of each execution of Par-CosNet. Data collected in this table is useful to assess the scalability of the parallelization over the

Table 5.3: Average COSNet performance in predicting GO protein functions.

Organism	Precision	Recall	F	AUPRC
Yeast - CC	0.2827	0.5252	0.3407	0.3611
Yeast - MF	0.2071	0.3705	0.2519	0.2513
Yeast - BP	0.2225	0.3765	0.2624	0.2628
Mouse - CC	0.1144	0.1790	0.1320	0.1172
Mouse - MF	0.0865	0.1464	0.1032	0.0908
Mouse - BP	0.0550	0.0796	0.0626	0.0504
Human - CC	0.1534	0.2564	0.1806	0.1631
Human - MF	0.0885	0.1399	0.1032	0.0889
Human - BP	0.0705	0.1126	0.0823	0.0684

Table 5.4: Average CPU time in seconds for COSNet and Par-COSNet to perform a CV cycle on a single GO term.

Method	Yeast	Mouse	Human
COSNet	8.86	107.57	84.03
Par-COSNet	0.28	1.71	1.49
Par-COSNet4	0.09	0.54	0.48
Par-COSNet8	0.07	0.33	0.31
Par-COSNet12	0.06	0.28	0.26

number of CPU threads. Optimal values for this table should be ideally near $100\% \times n$, with n the number of threads assigned to the task. As an example, optimal scalability for Par-COSNet executed on 12 CPU threads is achieved when occupancy reaches 1200%.

Finally, to evaluate also the memory usage, Table 5.7 reports the maximum memory footprint of COSNet and Par-COSNet when predicting a single GO term. For COSNet, memory usage excludes the memory required by the R interpreter itself, thus counting just the space of objects created by R and C procedures.

The time reduction obtained by Par-COSNet is impressive (Table 5.4): for instance on mouse data the execution time is reduced from $107s$ to $1.71s$, when using a single thread. Multithreading further accelerates the execution, passing to $0.54s$, $0.33s$ and $0.28s$ respectively when using 4, 8, and 12 CPU threads. To better understand these results, we report in Table 5.5 the speed-up gained by Par-COSNet when compared with COSNet. Even when using a single thread implementation, Par-COSNet achieves a speed-up

Table 5.5: Average speed-up for Par-COSNet to perform a CV cycle on a single GO term. COSNetis used as baseline for calculating the speed-up.

Method	**Yeast**	**Mouse**	**Human**
Par-COSNet	31.64x	62.90x	56.39x
Par-COSNet4	98.44x	199.20x	175.06x
Par-COSNet8	126.57x	325.96x	271.06x
Par-COSNet12	147.66x	384.18x	323.19x

Table 5.6: Average CPU occupancy in percentage for Par-COSNet to perform a CV cycle on every GO term. Optimal scalability is achieved when the occupancy reaches $100\% \times n$ with n the number of CPU threads.

Method	**Yeast**	**Mouse**	**Human**
Par-COSNet	99%	100%	100%
Par-COSNet4	345%	364%	368%
Par-COSNet8	613%	659%	692%
Par-COSNet12	815%	941%	975%

of range 31.64× (yeast) and 62.90× (mouse), whereas it gains up to two order of magnitude with respect to COSNet implementation when using in multithreaded version, up to 383.18× on mouse data.

Moreover, Par-COSNet occupancy is not far from the optimal value when using 4 threads, while slightly decreasing (in proportion) when the number of threads increases (see Table 5.6). This is due to the fact that all the CPU threads concurrently access the same GPU, creating a minor bottleneck in computation. Indeed, we are fairly sure that adding more GPUs to the host system significantly improves the occupancy in multithread execution, since the workload can be equally divided between the devices.

Although having been executed with 1, 4, 8 and 12 CPU threads, we only report the memory footprint of the single-thread execution, since the allocation footprints for the 4, 8 and 12 thread runs are pretty similar: as a matter of fact, maximum memory consumption is reached before the actual computation starts, i.e. while importing and compressing the network file (Table 5.7). Results show that, thanks to the compressed representation of both the net and labeling, memory usage is remarkably decreased, ranging from almost the half memory used by Par-COSNet on yeast, to less than one third on human and mouse.

Table 5.7: Maximum memory usage in GigaBytes for COSNet and Par-COSNet when running a CV cycle on a single GO term.

Method	**Yeast**	**Mouse**	**Human**
COSNet	0.40	3.73	3.26
Par-COSNet	0.27	1.13	0.94

Artificial data

Tests performed on the artificial dataset are aimed to evaluate the potential application of Par-COSNet on big data. We tested the method on the single AFP problem, hence the method has been executed in a single CPU thread mode. Table 5.8 shows the average execution time in seconds and the maximum memory occupancy in GigaBytes of Par-COSNet. Three dataset sizes have been considered, (each corresponding to a column in the table) and, for each size, datasets having different density (average degree per node) have been generated.

Table 5.8: Average CPU time in seconds (upper Table) and maximum memory consumption in GB (lower Table) for Par-COSNet to perform a single CV cycle on one class over the synthetic datasets.

Density	Execution time (s)		
	500k nodes	1000k nodes	1500k nodes
50	45.3	137	330
100	54.3	166	360
300	92.9	248	609

Density	Max memory allocation (GB)		
	500k nodes	1000k nodes	1500k nodes
50	1.94	3.86	5.81
100	3.69	7.37	11.1
300	10.7	21.4	32.1

Interestingly, Par-COSNet is able to predict node labels on graphs with 1.5 millions of nodes and average degree 300 in around 10 minutes, and using less than 32 GigaBytes of RAM, nowadays available on the majority of ordinary off-the-shelf workstations (Table 5.8). Furthermore, Par-COSNet shows a good scalability in terms of both computational time and memory consumption. The execution time increases less than linearly with the density, and little more than linearly with the number of nodes. The memory usage grows less than linearly with the number of nodes and little

more than linearly with the density. This is likely due to the fact that most of the memory consumption occurs with the import and compression of the net. Nevertheless this limitation can be addressed by off-line preprocessing the data and then importing the resulting file in compressed format. This strategy would allow Par-COSNet to process even larger datasets: as an example, the actual memory footprint recorded during the computation of the dataset composed by 1.5 million nodes with the average density of 300 edges per nodes, is around 4 GigaBytes of RAM memory and 1 GigaByte of GPU memory, which is a tiny fraction compared to the maximum quantity of RAM used for compressing the graph.

5.2 MCMC Colorer

For Par-COSNet we developed a parallel coloring method based on the Luby greedy algorithm. Despite this solution is effective in producing a proper coloring, generally minimizing the number of colors, it produces highly skewed color classes. This is undesirable for many applications, such as parallel job scheduling, that require balancing among classes. For this reason, we propose an alternative coloring strategy based on probabilistic methods, and expressively developed for parallel execution. We plan to incorporate this coloring strategy in a future version of Par-COSNet.

5.2.1 Background

The graph coloring problem consists in finding an assignment of colors to the vertices of a given graph such that no two adjacent vertices share the same color. As graph theory plays significant roles in modeling real-world problems and exhibits numerous applications in Pattern Recognition, Operation Research, Chemistry, Physics and Engineering disciplines among others, graph coloring finds very practical applications, for example, in social networks problem such as Community Identification in Dynamic Social Networks [343], summarization of social networks messages [344], Improving Friends Matching in Social Networks [345], and for Collective Spammer Detection [346]. Moreover, it can be used for recording medical or biometric images [347, 348] and for finding good resource allocation for device-to-device (D2D) communications [349, 350], used in modern wireless communication systems [351].

A common characteristic of these problems is that the graphs have very large size, thus requiring a speed up of the traditional greedy sequential coloring heuristics [352] typically by introducing parallelization techniques. For

instance, in [353] and [354] the authors focus on thread scalability for hundreds or thousands of threads, showing that it is possible to achieve both good performance and high quality with massive parallelism. Thus, thread scalability in parallel algorithms, namely the ability of hardware and software to deliver greater computational power when the amount of resources is increased, is crucial and widely exploited in this work.

In the literature, parallel graph coloring problem has been tackled by several approaches but, at the best of our knowledge, very few of them address the problem of balancing the color classes in parallel manner. One category of them is based on searching for a maximal independent set of vertices on a progressively shrunk graph and on the concurrent coloring of the vertices in the found independent set. Often the independent set itself is computed in parallel using some variant of the Luby's algorithm [339]. Examples of such approaches are [355, 356]. Another category includes methods that color as many vertices as possible concurrently, tentatively tolerating potential conflicts, while detecting and solving conflicts afterwards (e.g. [357]). Despite these solutions are effective in producing a proper coloring, generally minimizing the number of colors, they produce highly skewed color classes, undesirable for many applications, such as parallel job scheduling, that requires balancing among the classes. At the other extreme, one could search for a coloring being *equitable*, that is a coloring that guarantees that the sizes of any two color classes differ by at most one [358]. This constraint is very expensive and somehow too stringent for practical applications; moreover class size constraints are known to usually raise computational hardness when performing partitioning [359]. *Balanced* coloring relaxes the equitable constraint requiring that any two color class sizes differ by an integer l greater than 1. Few approaches have been proposed to tackle Balanced graph coloring (e.g. [356, 360]). However, the limit of these methods is still that they are intrinsically sequential thus not scalable, becoming unfeasible on large graph. A promising direction of research on graph coloring concerns the Markov Chain Monte Carlo (MCMC) methods that allow sampling from non analytic complex distributions. The idea is to define an ergodic Markov chain whose steady state distribution is defined over the set of colorings we wish to sample from. Within the framework of graph coloring using Markov chains several contributions have been proposed. In [361] a simple sequential solution based on the Glauber dynamics has been adopted. The Glauber dynamics produces a Markov chain on a proper coloring where at each step a random vertex v is recolored, choosing a color uniformly at random from the permissible ones.

To deal with large graphs, we present an algorithm based on MCMC method producing balanced graph coloring in a parallel way. Moreover, we

show the effectiveness of this stochastic approach through experiments on random and real-world graphs. Briefly, in this work we analyze the convergence the proposed MCMC algorithm on the basis of the probability rules used in the different stages of the overall sampling process; we provide an intuitive and non-rigorous analysis about the balancing mechanism, whereas we assess the quality of the final balancing through a suitable asymptotic hypothesis test of statistical fitting; we have conducted experiments on randomly generated and social graphs (extracted from real data).

As for experiments, to achieve significant speedups we leverage on modern many-core GPUs architectures [332]. Numerical results also show that the number of threads used by the parallel algorithm scales well with the graph sizes, even if compared with the GPU exploitation by the greedy strategy.

5.2.2 Material and methods

Notations

We will consider a simple undirected graph $\mathcal{G} = \langle V, E \rangle$ with $n = |V|$ vertices and the set $[k] = \{1, \ldots, k\}$ of colors used to label the vertices. A k-coloring $c : V \to [k]$, also represented as vector $c = (c(1), \ldots, c(n)) \in [k]^n$, is called *proper* if adjacent vertices receive different colors, otherwise it is termed *improper*. It is well-known that, if $\Delta(\mathcal{G})$ is the maximum degree of $\mathcal{G}$, $k = \Delta(\mathcal{G}) + 1$ colors are sufficient to properly color the graph by a sequential greedy algorithm.

For a given coloring c, let $\mathcal{N}(v)$ denote the neighborhood of node v in $\mathcal{G}$, and $c_{\mathcal{N}(v)} \subseteq [k]$ be the set of colors occupied by vertices $\mathcal{N}(v)$ and $\bar{c}_{\mathcal{N}(v)}$ its complement; let us denote the respective cardinalities with $g_v(c) = |c_{\mathcal{N}(v)}|$ and $\bar{g}_v(c) = |\bar{c}_{\mathcal{N}(v)}|$. Given c, an edge $uv \in E$ with $c_u = c_v$ is a *conflict* and $\#(c) : [k]^n \to \mathbb{N}$ counts the number of conflicts. We will also consider the absolute *frequency* of the color j in c: $f_j(c) = |\{u \in V : c_u = j\}|$.

Hereafter we will use lowercase letters, e.g. c, c', c^*, for given colorings and uppercase for random colorings, e.g. C, C', C^*. For example the probability of $C' = c'$ given $C = c$ will be denoted $p(C' = c' \mid C = c)$ or $p(c' \mid c)$ for short.

Markov Chain Monte Carlo for sampling colorings

This novel parallel Markov Chain Monte Carlo (MCMC) technique relies on a 1st-order ergodic Markov chain $(C^i)_{i=1}^{\infty}$ visiting a sequence of (possibly improper) k-colorings $c \in [k]^n$ and whose stationary distribution π strongly depends on the set of conflicts involved in c. As usual, we consider the *Gibbs*

distribution as target stationary distribution for the Markov chain:

$$\theta(c) = \frac{e^{-\beta \#(c)}}{Z(\beta)}, \qquad \text{with } Z(\beta) = \sum_{c' \in [k]^n} e^{-\beta \#(c')}. \tag{5.5}$$

The role of parameter β in (5.5) aims to penalize improper colorings, leading π toward the uniform distribution over the proper colorings only with exponential decrease (as β increases). It is known from MCMC theory that the latter distribution is asymptotically approached in the sampling process when the chain is suitably constructed using the well-established Metropolis-Hastings algorithm [362]. This construction requires the specification of a *proposal* probability encapsulating the *acceptance ratio* and a *transition* probability for the chain.

As for the transition probabilities of the Markov chain, given a coloring $C = c$ we sample the successive coloring C^* in two phases acting according to a typical "rejection sampling" scheme [363]: first a candidate coloring C' is generated according to a suitable proposal probability $r(c, c') := p(c' \mid c)$, then the proposal C' is accepted effectively as successive coloring C^* according to the acceptance ratio $\alpha(c, c') := \min\left\{\frac{\theta(c')r(c',c)}{\theta(c)r(c,c')}, 1\right\}$:

$$p(c' \mid c) := \begin{cases} \alpha(c, c'), & c' \neq c \\ 1 - \alpha(c, c'), & \text{otherwise} \end{cases}.$$

The proposal coloring C' is sampled with probability $r(c, c')$ as follows. Each node $v \in V$ is drawn independently and with identical distribution $p(c'_v \mid c)$ of colors so that the overall proposal probability is

$$r(c, c') = \prod_{v \in V} p(c'_v \mid c). \tag{5.6}$$

Notice that, in the construction of the acceptance ratio also the *backward* probability $r(c', c)$ is required, hence $r(c, c')$ is called *forward* probability.

The choice of the node proposal probability $p(c'_v \mid c)$ is a key step and is hence detailed distinctly in the following subsection. It is also important to observe from the computational viewpoint that the independent drawing of all c'_v, $v \in V$, allows for the generation of the new coloring in a parallel manner.

Proposal distribution for new colorings

Here we specify the algorithm for the proposal distribution procedure so that the stochastic evaluations follow consequently from the analysis of the

color generation. First, the behavior of the algorithm splits into two cases based on the old coloring c. When there is some conflict locally for v, namely $c_v \in c_{\mathcal{N}(v)}$, the new proposed color C'_v for v shall be redrawn with the aim of reducing the possible conflicts. We draw it from the *free* colors $\bar{c}_{\mathcal{N}(v)}$ following a nearly uniform distribution of $C'_v = j$ given c:

$$\eta_v(j, c) = \begin{cases} \frac{1-\varepsilon\, g_v(c)}{k-g_v(c)}, & \text{if } j \in \bar{c}_{\mathcal{N}(v)} \\ \varepsilon, & \text{if } j \in c_{\mathcal{N}(v)}. \end{cases} \tag{5.7}$$

The rationale behind such definition is that we want to generate with high probability, a color equally likely among those free, in order to aim at the balancing objective of the method. Nevertheless, we keep a negligible chance $\varepsilon > 0$ to pick a color that is not free, in order to widen the search space.

As for the case of no conflict for v, i.e. $c_v \in \bar{c}_{\mathcal{N}(v)}$, it is desirable to keep nearly surely the old color c_v to facilitate the convergence of the algorithm, or otherwise pick another color with a small chance ε. Hence, in the case of no conflict, for the node v the proposal color $C'_v = j$ given c is distributed as

$$\zeta_v(j, c) = \begin{cases} 1 - \varepsilon\,(k-1), & \text{if } j = c_v \\ \varepsilon, & \text{if } j \neq c_v. \end{cases} \tag{5.8}$$

With the above definitions we derive the following conditional distribution for proposal color (also called forward):

$$p(c'_v \mid c) = \begin{cases} \eta_v(c'_v, c), & \text{if } g_v(c) < k, c_v \in c_{\mathcal{N}(v)} \\ \zeta_v(c'_v, c), & \text{otherwise.} \end{cases} \tag{5.9}$$

The backward probabilities

$$r(c', c) = p(C' = c \mid C = c') = \prod_{v \in V} p(C'_v = c_v \mid C = c')$$

can be obtained by symmetrical reasoning, i.e. exchanging the role of c and c' in the calculations outlined above. This allows to compute then the acceptance ratio $\alpha(c, c')$.

The main procedural steps of the MCMC algorithm described above are sketched in Algorithm 6.

Algorithm analysis

This section deals with the study of the behavior of the highly scalable parallel MCMC algorithm for graph coloring outlined in the previous section.

Algorithm 6 Parallel MCMC Graph Coloring

Input: Graph $\mathcal{G} = \langle V, E \rangle$ with $n = |V|$;
Number k of colors;
Gibbs parameter $\beta \ll 1$
Output: Random proper coloring $C \in [k]^n$

$C \leftarrow$ random initial coloring $\in [k]^n$
while $\#(C) > 0$ **do**
  for $v \in V$ **in parallel do**
    Calculate $C_{\mathcal{N}(v)}$
    $g_v(C) \leftarrow |C_{\mathcal{N}(v)}|$
    $p(c'_v \mid C) \leftarrow$ Compute according to (5.9)
    $C'_v \leftarrow$ Generate with distribution $p(c'_v \mid C)$
  end for

  $C' \leftarrow$ Proposed coloring $(C'_1, C'_2, ..., C_n)$
  $r(C, C') \leftarrow$ Compute forward probability
  $r(C', C) \leftarrow$ Compute backward probability
  $\alpha(C, C') \leftarrow \min\left\{\frac{r(C',C)}{r(C,C')} e^{-\beta(\#(C')-\#(C))}, 1\right\}$
  Accept $C \leftarrow C'$ with probability $\alpha(C, C')$
end while

First, we claim that in the Markov chain, the color distribution stochastically converges to proper colorings by repeatedly sampling a pool of easier proposal distributions built upon the sets of neighboring node colors. Then, we also develop a discussion about the quality of balancing achieved, the latter being the main goal of this modeling.

As for the convergence properties of Algorithm 6, due to the intrinsic randomness in drawing colors we cannot guarantee that the number of conflicts $\#C$ strictly decreases at each iteration. Therefore we give a characterization of the convergence in probabilistic terms, studying a slight variant of Algorithm 6 where we bring the parameters to the extreme values: acceptance ratio $\alpha(C, C') = 1$ and $\varepsilon = 0$.

Lemma 1. *Given current coloring C, let $\#T$ and $\#T^*$ be number of nodes in $\mathcal{G}$ having some conflict at, respectively, any current and successive iteration of Algorithm 6, where we set acceptance ratio $\alpha(C, C') = 1$ and $\varepsilon = 0$. For any integer $r > 1$ and provided a number of colors $k \geq \Delta(\mathcal{G}) + r$, the following*

expectation inequality holds:

$$\mathbb{E}[\#T^* \mid C] \leq \rho \#T$$

for some $0 < \rho = 1 - (1 - 1/r)^{\Delta(\mathcal{G})} < 1$.

Proof. Preliminarily, we introduce the convenient notation of *indicator* expression $\mathbb{1}(cond)$ that takes value 1 if the condition *cond* is true, and 0 otherwise. The number of conflicts in the new coloring C^* can be written as $\#C^* = \sum_{uv \in E} \mathbb{1}(C^*_u = C^*_v)$, while the number of *local* conflicts is $\#_v C^* = \sum_{u:uv \in E} \mathbb{1}(C^*_u = C^*_v)$; similar definitions can be given for $\#C$ and $\#C_v$. Given the current coloring C we partition V into nodes with conflicts, $T = \{u \in V : \#_u C > 0\}$, and conflict-free nodes $\bar{T} = V \setminus T$.

First, notice that the number of local conflicts at any $v \in T$ is simply

$$\#_v C^* = \sum_{u \in T : uv \in E} (C^*_u = C^*_v)$$

due to $\sum_{u \in \bar{T} : uv \in E} \mathbb{1}(C^*_u = C^*_v) = 0$, since when u is not in T it maintains the old color $C^*_u = C_u \neq C^*_v$. Hence, in the summation in $\#_v C^*$ each term $\mathbb{1}(C^*_u = C^*_v)$ is a Bernoullian variable with parameter

$$p_{uv} := \frac{|\bar{C}_{\mathcal{N}(u)} \cap \bar{C}_{\mathcal{N}(v)}|}{|\bar{C}_{\mathcal{N}(u)}| \cdot |\bar{C}_{\mathcal{N}(v)}|} \leq \frac{\min\left\{|\bar{C}_{\mathcal{N}(u)}|, |\bar{C}_{\mathcal{N}(v)}|\right\}}{|\bar{C}_{\mathcal{N}(u)}| \cdot |\bar{C}_{\mathcal{N}(v)}|} \leq \frac{1}{r},$$

conditioned on C, i.e.

$$\mathbb{1}(C^*_u = C^*_v) \mid C \sim \text{Bernoulli}(p_{uv}).$$

Moreover, these terms are stochastically independent since the drawing of C^*_u for each neighbor u of v is done autonomously from the other neighbors. Given C, the variable $\#_v C^*$ is hence a sum of independent Bernoullian variables having parameters p_{uv} with $uv \in E, u \in T$, namely a so-called Poisson-Binomial random variable:

$$\#_v C^* \mid C \sim \text{PoissonBinomial}(\{p_{uv} : uv \in E, u \in T\})$$

We define the Bernoulli variable $B^*_v := \mathbb{1}(\#_v C^* > 0)$ indicating whether v has some conflict in the new coloring. Thanks to the observation above, its distribution conditioned on C is easily determined as

$$(B^*_v \mid C) = (\mathbb{1}(\#_v C^* > 0) \mid C) \sim \text{Bernoulli}(1 - \Pi_{u \in W : uv \in E}(1 - p_{uv})) \quad (5.10)$$

when $v \in W$, since $\Pi_{u \in T : uv \in E}(1 - p_{uv})$ is the probability that v does not have any conflict with the neighbors in the new coloring C^*. Clearly, $B^*_v = 0$ if

$v \notin T$. The number $\#T^* = \#\{u \in V : \#_u C^* > 0\} = \sum_{v \in T} B_v^*$ of conflicting nodes in the new coloring C^* thus satisfies the following relationships

$$\mathbb{E}[\#T^* \mid C] = \mathbb{E}\left[\sum_{v \in T} B_v^* \mid C\right] = \sum_{v \in T} \mathbb{E}[B_v^* \mid C] \tag{5.11}$$

$$= \sum_{v \in T} \left[1 - \prod_{u \in T : uv \in E} (1 - p_{uv})\right] \tag{5.12}$$

$$\leq \sum_{v \in T} \left[1 - \prod_{u \in T : uv \in E} \left(1 - \frac{1}{r}\right)\right] \tag{5.13}$$

$$\leq \sum_{v \in T} \left[1 - \left(1 - \frac{1}{r}\right)^{\deg v}\right] \leq \left[1 - \left(1 - \frac{1}{r}\right)^{\Delta(\mathcal{G})}\right] \sum_{v \in T} B_v \tag{5.14}$$

$$= \left[1 - \left(1 - \frac{1}{r}\right)^{\Delta(\mathcal{G})}\right] \#T = \rho \#T \tag{5.15}$$

where $0 < \rho := 1 - (1 - 1/r)^{\Delta(\mathcal{G})} < 1$. That is, the claim $\mathbb{E}[\#T^* \mid C] \leq \rho \#T$ holds. □

Directly following the previous result, we have the following convergence.

Theorem 1. *The number of conflicts $\#C$ converges in probability to 0, i.e.* $\lim_{t \to \infty} p(\#(C^t) < \delta) = 1 \ \forall \delta > 0$.

Proof. From Lemma 1, using the monotonicity property of the expected value we have

$$\mathbb{E}[\mathbb{E}[\#T^* \mid C]] \leq \rho \mathbb{E}[\#T].$$

By the Tower Rule of the conditional expectation ($\mathbb{E}[\mathbb{E}[\#T^* \mid C]] = \mathbb{E}[\#T^*]$) we can give the bound

$$\mathbb{E}[\#T^*] \leq \rho \mathbb{E}[\#T].$$

The coloring C^* is the one successive to C; hence building the sequence of successive colorings C^t, $t = 0, 1, 2, ...$ by means of the algorithm above, one can guarantee $\mathbb{E}[\#T^t] \leq \rho^t \mathbb{E}[\#T^0]$ for any t. Hence, applying $\lim_{t \to \infty}$ to both side, we have the convergence in expectation of the number of conflicting nodes to 0:

$$\lim_{t \to \infty} \mathbb{E}[\#T^t] = \lim_{t \to \infty} \rho^t \mathbb{E}[\#T^0] = 0.$$

Now, it is well known that convergence in expectation (namely, L^1-convergence in the proper probability space $(\Omega, \mathcal{F}, p)$) implies the convergence in probability, i.e.

$$\lim_{t \to \infty} p(\#T^t < \delta) = 1 \qquad \text{for any } \delta > 0.$$

Since the number $\#C^t$ of conflicts is easily bounded above by $\frac{1}{2}\Delta(\mathcal{G})\#T^t$, it converges in probability to 0 as well:

$$\lim_{t\to\infty} p(\#C^t < \delta) = 1 \qquad \text{for any } \delta > 0.$$

□

The balancing of color classes is an important aspect but it is difficult to prove it rigorously, due to the complex role that the distributions arising from the model play in the evolution of the algorithm towards attaining the final coloring. Nevertheless, in this section we sketch a qualitative non-rigorous analysis of this process in order to provide some intuition and rationale about the nearly uniform character of final color distributions, regardless of the topology of the graphs at hand.

To carry out this analysis, we resort to the (n, k)-Poisson Multinomial Distribution (PMD) [364] which is the distribution of the sum of n independent random vectors supported on the set $\mathcal{B}_k = \{e_1, \ldots, e_k\}$ of standard basis vectors in $\mathbb{R}^k$, representing the k colors provided to the algorithm. We replace the representation of the color set $[k]$ with $\mathcal{B}_k$ for technical reasons in this section only.

At each iteration step t of the Algorithm 6, each node $v \in V$ randomly draws a color with probability distribution (5.9). Let $F_v^{(t)} \subseteq \mathcal{B}_k$ be the set of free colors, at step t, for node v whose color is represented by the random variable $C_v^{(t)} \in \mathcal{B}_k$, and let $p_v^{(t)} = \left(p_{v,1}^{(t)}, \ldots, p_{v,k}^{(t)}\right)$ be the distribution of the free colors for v on $\mathcal{B}_k$. Clearly, due to the random drawing triggered by (5.9) we have

$$p_{v,j}^{(t)} \approx \begin{cases} 1/|F_v^{(t)}|, & \text{if } e_j \in F_v^{(t)} \\ 0, & \text{if } e_j \notin F_v^{(t)} \end{cases} \tag{5.16}$$

in case of conflict for v (eq. (5.7)), or

$$p_{v,j}^{(t)} \approx \begin{cases} 1, & \text{if } e_j = C_v^{(t)} \\ 0, & \text{otherwise} \end{cases}, \tag{5.17}$$

when v is conflict-free (eq. (5.8)). This means that the process of assigning a new color $C_v^{(t+1)} \in \mathcal{B}_k$ to node v, can be expressed by a multinoulli distribution with parameter $p_v^{(t)}$, i.e.,

$$C_v^{(t+1)} \sim \text{Multinoulli}(p_v^{(t)}).$$

Notice that each node in the color $C_v^{(t+1)}$ is drawn independently from each other.

Let $X^{(t)} = \sum_{v \in V} C_v^{(t)}$ be the number of occurrences of all colors in the graph coloring at time t. In particular, for each $j \in [k]$, $X_j^{(t)}$ represents the frequency of color e_j within the coloring $C^{(t)}$. Hence, the random vector $X^{(t)}$ is the sum of independent but not identically distributed multinoulli random variables, which follows a (n, k)-PMD, where n is the number of vertices and k the number of colors. Let $\xi(x) = p(X^{(t)} = x)$ denote the probability mass function of $X^{(t)}$, where x is a k-dimensional vector with non-negative integer entries summing up to n. Clearly, in order to have the desired balancing, it is crucial to assign high probability $\xi(x)$ to the elements x with equalized entries. In the following, we provide a qualitative and non-rigorous analysis of the dynamics of color assignments and its distribution evolution in order to support the experimental evidence that the color classes are quite balanced among each other. Let us investigate the following steps in the coloring process generated by Algorithm 1.

Step 1. Assume that in the first step the colors are assigned independently and uniformly at random at each node. As a consequence, the set of free colors $F_v^{(1)}$ of v comes up uniformly among all subsets of size $\left|F_v^{(1)}\right|$. Let us observe how a new coloring is generated by focusing on a color e_j through all the n independent trials, each one corresponding to a node v. To consider the overall chances e_j has, we have to focus in turn on the probability sets $P_j = \left\{p_{v,j}^{(1)} : v \in V\right\}$. In particular, for large n and under the above independence assumptions, it results that $p_{v,j}^{(1)}$ is the same on average for all v due to the uniformness of $F_v^{(1)}$, so that e_j is assigned nearly the same chance to be selected in each trial. In other words, the averages over all nodes $\bar{P}_j$ should be approximately $1/k$ for large graph size n.

Step t. Assuming that at time $t-1$ the (in general improper) coloring $C^{(t-1)}$ is quite balanced in terms of color frequencies, we can conclude that also the new coloring $C^{(t)}$ could achieve a good balancing. This should happen thanks to the following two facts. First, some colors are definitely fixed since they are randomly chosen in some previous step, as described by the (5.17) and this contribute to uniformly spread the colors in the neighborhoods of those nodes that have to make a choice. Second, those nodes that have to update the color, are in the same condition described in Step 1, thus they repeat the same independent drawings.

Figure 5.3 provides an empirical demonstration of the balancing trend exhibited by the MCMC algorithm and described in the above steps for a

social graph (labeled *ca-coauthors-dblp*) introduced in the next section on numerical simulations. In fact, the balancing quality in each repetition (light blue lines) and on average (red line) is maintained quite constant over all iterations (**Step t**) apart from the first iteration (**Step 1**) that requires an adjustment as a consequence of the random initial coloring.

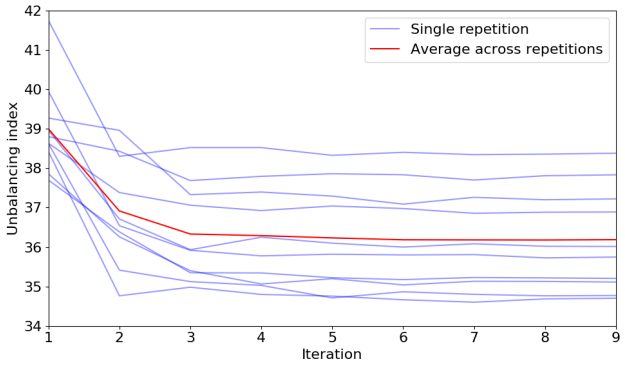


Figure 5.3: Color balancing quality of a sample graph (the social graph *ca-coauthors-dblp*) during the iterations of the parallel MCMC algorithm for the first 10 states (colorings) visited by the Markov chain.

5.2.3 Results and discussion

In this section we report some numerical simulation results of the parallel MCMC method. Our coloring strategy has been evaluated on artificial / randomly generated graphs and on real-world graphs in the context of social network.

To this aim, we developed a fast parallel implementation of the presented strategy, hereafter called *MCMC-GPU*, using the NVIDIA CUDA programming paradigm. NVIDIA GPU processors feature up to 5000 processing cores, hence a very large number of processing threads can be scheduled and executed concurrently in a shared memory model. Thanks to this, parallelization in MCMC-GPU occurs at vertex level, i.e. a thread is assigned to each vertex of the graph which is therefore processed concurrently to every other vertex.

MCMC-GPU implementation closely follows Algorithm 6: during the iterations, each vertex v is assigned to a processing thread which evaluates both $p(c'_v \mid c)$ and $p(c_v \mid c')$ (the forward and backward probabilities) and draws the new color c_v accordingly. Thread synchronization occurs only at the end of each iteration, where the total number of conflicts and the rejection

factor $\alpha(c, c')$ of the new coloring have to be evaluated. In MCMC-GPU the algorithm is executed until a proper k-coloring is found, or the maximum number of allowed iterations is reached.

We compared our GPU implementation to a non-parallel implementation (called *MCMC-CPU*) of the same algorithm which runs on a standard CPU. The purpose of this comparison is to evaluate the scalability and speed-up of the parallel approach taking the sequential implementation as baseline reference.

Finally, as a third comparison, we tested the MCMC-GPU implementation against a fast and fully parallel greedy coloring strategy inspired by [339] work, called *Luby-GPU*. Luby has described a greedy parallel strategy to find a maximal independent set (MIS) of vertices (i.e. a subset of vertices such that no two vertices are neighbors) in undirected graphs. Consequently, given that any MIS can be colored in parallel, a greedy graph coloring strategy could be defined by repeatedly finding the largest MIS on subgraphs gradually resulting from pruning previous recovered MIS. Clearly, the Luby inspired colorer is not meant for balanced graph coloring problem. However, we use it for two reasons: on one hand just for sake of comparison with a simple scheme graph colorer, on the other hand to empirically show that the two algorithms have comparable computational times on sparse graphs.

All the algorithms were implemented in C++ and both MCMC-GPU and Luby-GPU levarages the NVidia CUDA programming paradigm ([332])[1] All the host code is single-threaded and it is meant to be run on a single machine, hence no other libraries (such as pthread, C++11 threads, OpenMP, MPI, etc...) are required. Both CPU and GPU code is compiled with NVidia nvcc v10.1, using the underlying GNU GCC v5.4.0 as C++ compiler.

For running the experiment we used a workstation with 2× Intel Xeon CPU E5-2620 v3 @ 2.40GHz, 64GB of RAM, Linux 16.04 and an NVidia GTX980 GPU featuring 2880 cores and 4GB of VRAM. MCMC-CPU runs are single-threaded, hence were run on a single core of the host CPU. For LubyGPU and MCMC-GPU, on the host the processes are single-threaded as well, and parallelization occurs only on the GPU side where 2880 cores of a single NVidia GTX980 GPU are used.

In all tests we aim at measuring the quality of balancing in the color class sizes produced by all the considered coloring strategies, in particular by MCMC-GPU, as this works by improving the balancing of an existing (improper) coloring moving vertices from one color class to another on the basis of random choices, as highlighted by the proposed distribution over

[1]The software used for the experiments is freely available on the GitHub repository: `https://github.com/phuselab/MCMC_Colorer`.

colors given in Section 5.2.2. Let us first introduce a measure of deviation of a coloring $c \in [k]^n$ from a perfectly balanced coloring where each color class j has size $n_j = n/k$, for each $j \in [k]$. This can be quantified by defining $\gamma_{n,k}(j) = |f_j - n/k|$ which represents the target to be minimized in respect of which our heuristic will perform local random arrangements. An overall measure of balancing quality closely related to the standard deviation of the color class sizes can be then defined as

$$\Gamma_{n,k}(c) = \left(\frac{1}{k} \sum_{j=1}^{k} \gamma_{n,k}^2(j) \right)^{1/2} \tag{5.18}$$

called *unbalancing index* hereafter. Clearly, a coloring c is perfectly balanced if $\Gamma_{n,k}(c) = 0$.

Artificial Data

Here we report some numerical simulation results of the parallel MCMC method exploiting the Erdős-Rènyi graph (ER) model [365]. This model is widely used to generate random graphs and has some valuable properties to leverage in order to asses the behavior of the proposed coloring strategy both for fixed graph sizes and asymptotically.

In the ER model $\mathcal{G}(n,p)$, a n-vertex graph is constructed by connecting vertices randomly and including each edge with probability p independently from every other edge. Equivalently, the probability that a vertex v has degree k is Binomial, i.e. $p(\deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$, with expected value $\mathbb{E}[\deg(v)] = (n-1)p$. As n goes to infinity, the probability that a graph in $\mathcal{G}(n, 2\ln(n)/n)$ is connected, tends to one. Another relevant property of ER graphs is the edge density which is a random variable with expectation exactly equal to the background probability p.

The role that ER model plays in this picture is important because it ensures the possibility to provide graphs with certain properties, giving us at same time an effective and sound algorithmic procedure to compute them in practice.

A demonstration on how MCMC-GPU and Luby-GPU perform in terms of color balancing is given in Fig. 5.4, where the curves represent the unbalancing index (5.18).

More precisely, the graphics relate to average amounts achieved on ER graphs of various size and densities 0.1% and 0.5% respectively. To capture a wide scale of ER graphs, once the density p has been fixed, we varied the graph size n up to 500K vertices, averaging over 10 trials for each pair (n,p) to reach satisfactory confidence level.

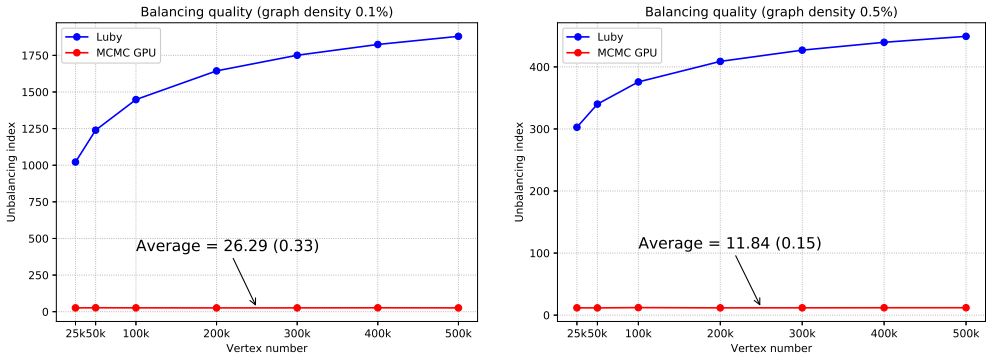


Figure 5.4: Average unbalancing index achieved by MCMC-GPU and Luby-GPU on ER graphs of various size and densities 0.1% and 0.5% respectively.

Regarding the number of colors k used by MCMC-GPU, assuming a regular structure of the generated graphs we fix $k = \lceil np \rceil$ corresponding to the expected vertex degree, which anyway assures the existence of proper coloring with high probability. Note that, under this setting a coloring c for a graph in $\mathcal{G}(n,p)$ has balancing index

$$\Gamma_{n,\lceil np \rceil}(c) \approx \left(\frac{1}{np} \sum_{j=1}^{k} \gamma_{n,k}^2(j) \right)^{1/2}.$$

As can be noticed in the plots, MCMC-GPU not only outperforms Luby-GPU, but also provides invariant unbalancing index with respect to the graph sizes, being $\Gamma_{n,\lceil np \rceil}(c) \approx$ constant (with very low standard deviation), for all n in the range 25K $\div$ 500K. In spite of the limited number of experiments, as a general trend we have that the sum of within-class deviations $\gamma_{n,k}^2(j)$ roughly grows linearly with the number of vertices, i.e. $\sum_{j=1}^{k} \gamma_{n,k}^2(j) \approx pc_p n$, where c_p is a constant depending on the density p. For instance, in Fig. 5.4, the constants $c_p = 26.29$ and $c_p = 11.84$ are shown for $p = 0.1\%$ and $p = 0.5\%$ respectively.

With regard to the computational times of the conducted experiments, their averages are reported in Fig. 5.5. Whereas we can notice very high speedups (up to 20) between sequential and parallel MCMC implementations, the times spent by MCMC-GPU and Luby-GPU remain comparable (they turn in favor of Luby-GPU only for 500K).

A second experiment is aimed at studying the balancing quality achieved when varying both the graph density and the number of colors made available to MCMC-GPU. In particular, here we set the graph density in terms of vertex degree $d = \lceil np \rceil$, with d falling in the range 100 $\div$ 350, while the

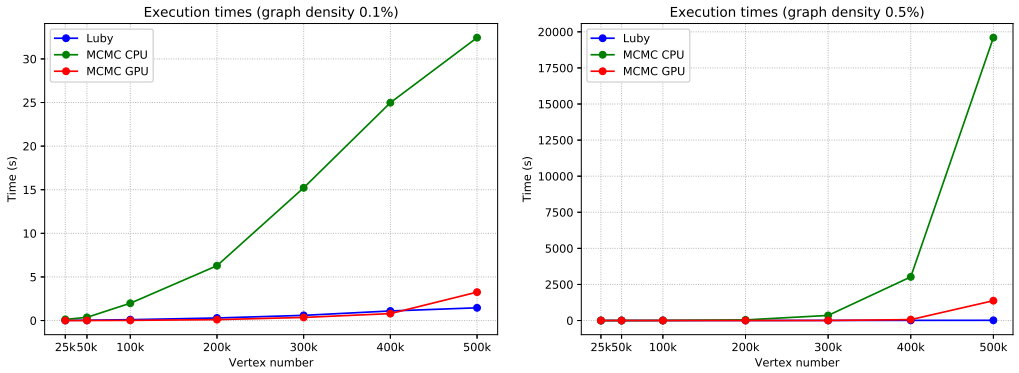


Figure 5.5: Average execution times of Luby-GPU, MCMC-CPU and MCMC-GPU on ER graphs of various size and densities 0.1% and 0.5% respectively.

number $k = r\lceil np \rceil$ of colors is scaled down by a factor $r \in (0, 1)$ ranging from 0.5 and 1. Average values of the unbalancing index over colorings carried out by the two algorithms are plotted in Fig. 5.6 (note that the ratio between the scales of the two graphs is about 33).

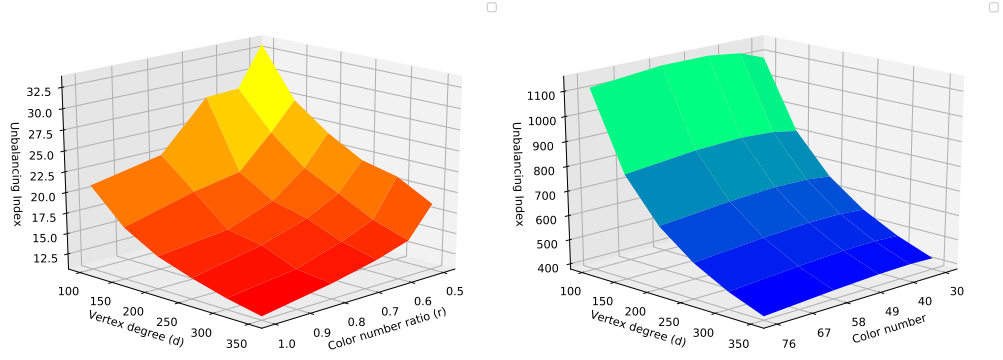


Figure 5.6: Average unbalancing index achieved by MCMC-GPU (left) and Luby-GPU (right) on ER graphs varying both vertex degrees $d = \lceil np \rceil$ and color number k.

Real Data

Here we extend the evaluation to a small sample of real-world instances, where graphs describe relations among individuals, to identify differences and analogies in behaviour between synthetic and real-world data. A second purpose of these experiments is to empirically validate both algorithm convergence and balancing properties in the light of the analysis delineated in

the previous section.

The three coloring approaches have been tested on four real-world social graphs taken from the [366] repository. They greatly vary in terms of size (number of nodes and edges) and average/maximum degree, thus providing a small but relatively diverse array of use cases. Their relevant features are summarized in Table 5.9. The maximum number of possible colors to be used is a parameter in our algorithm and it has to be fixed in advance. We had fixed this parameter in such a way that the results are stable for each trail. Table 5.9 shows the values of this parameter for each graph used in the experiment.

Table 5.9: Features of the graphs used in the experiments: In the upper table, the number of nodes and edges (columns 2-3), maximum and average degree (columns 4-5); in the lower table, the average number of colors found by Luby-GPU (column 2) and the parameter of maximum number of colors to be used, set for MCMC-GPU (and MCMC-CPU) (column 3).

Graph	**Nodes**	**Edges**	**Max deg**	**Avg deg**
sc-shipsec5	179.1K	4.4M	75	24
sc-pwtk	218K	11.4M	179	51
ca-coauthors-dblp	540K	30M	3.3K	56
ca-hollywood-2009	1.1M	56M	11.5K	105

Graph	**N. colors by Luby**	**N. colors by MCMC-GPU**
sc-shipsec5	33	50
sc-pwtk	43	85
ca-coauthors-dblp	337	460
ca-hollywood-2009	2209	2400

Figure 5.7 top shows the average unbalancing index over 35 repetitions for each coloring algorithm applied to the 4 sample graphs. Giving to its greedy design, Luby-GPU achieved a worse balancing performance, being largely outperformed by MCMC strategy. For the latter we set a number of colors barely greater than to that obtained by Luby (for details see Table 5.9).

Figure 5.7 bottom shows the average computing times over the trials. Being inherently sequential, MCMC-CPU is the slowest and a direct comparison with MCMC-GPU results in an average speed-up ranging from 28× (for *sc-pwtk*) up to 69× (for *ca-hollywood-2009*). We also remark that in

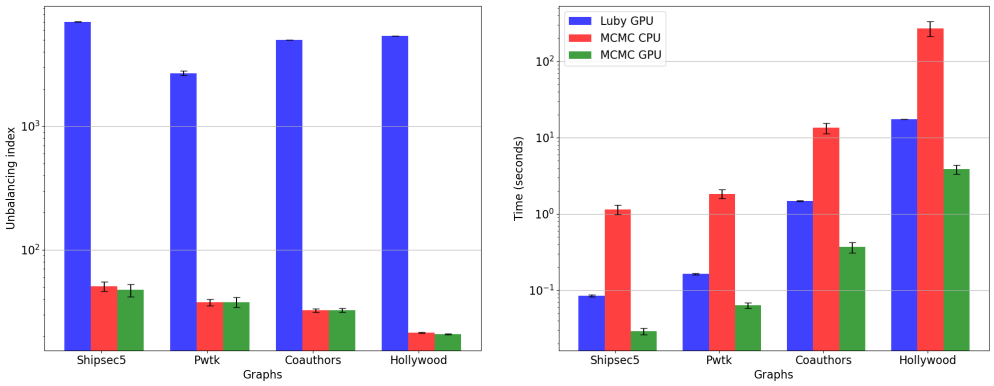


Figure 5.7: Average unbalancing index (left plot) of final colorings achieved by the three algorithms for the graphs of Table 5.9 and related average execution times (right plot), expressed in seconds. Results are on 35 repetitions (standard deviation as error bar) and plotted in logarithmic scale.

every experiment, MCMC-GPU produces a proper coloring for each graph in less time than Luby-GPU does, showing that sometimes it is even faster than the greedy approach.

For statistically assessing the balancing, we test the uniformness of color distribution through a hypothesis test starting from the balancing index (5.18). Given a coloring c produced by MCMC-GPU, we consider the statistic defined as

$$K^2 = \frac{(k\,\Gamma_{n,k}(c))^2}{n} = \sum_{j=1}^{k} \frac{(n_j - n/k)^2}{n/k}.$$

In fact, this expression is well known in statistical inference as the Pearson's chi-squared statistic for goodness-of-fit test [367, §IX.5.2], that is used for assessing whether the observed categorical outcomes of the experiment have frequencies (n_j) following a hypothesized category distribution, namely uniform distribution in our case (i.e. color class sizes n/k). With this so-called null hypothesis H_0, it was proved using the Central Limit Theorem and a suitable geometric transformation (see [368]) that such statistic K^2 has asymptotically a $\chi^2(k-1)$ distribution (chi-squared distribution with $k-1$ degrees of freedom) as $n \to \infty$: hence, $\chi^2(k-1)$ is a good approximation of the true distribution for large number n of nodes, as in our experiments.

The key point is that the statistic K^2 is large and the corresponding p-value is small, when the produced color class sizes are unbalanced, and vice-versa. Results of the test are reported in Table 5.10 for 35 runs: notice that the range of rather large p-values leads to accept the H_0 hypothesis stating the color balancing in all analyzed graphs.

Table 5.10: Test for fitting uniform color distribution on the result of 35 runs with the analyzed graphs. Ranges of observed Pearson's chi-squared test-statistics and corresponding p-values are shown.

Graph	K^2 **statistic range**	p**-value range**
sc-shipsec5	[19.51, 50.52]	[0.413, 0.999]
sc-pwtk	[31.14, 64.57]	[0.940, 1.000]
ca-coauthors-dblp	[368.09, 473.42]	[0.310, 0.998]
ca-hollywood-2009	[2205.43, 2446.97]	[0.242, 0.997]

5.3 Conclusions

Par-COSNet is a method for node label prediction in big graphs, well-suited to process large protein networks with strongly unbalanced labels. We presented applications to the protein function prediction problem, but it can be applied to other relevant problems in Computational Biology and Network Medicine, including disease gene prioritization and drug repurposing. Par-COSNet introduces a parallel and sparse implementation of COSNet, a state-of-art imbalance-aware method for predicting protein function, which allows to remarkably speed up the computation and reduce the memory requirements. In particular, the dynamics of the Hopfield network on which COSNet builds upon is parallelized by solving a vertex coloring problem on the graph/network, partitioning nodes into sets of independent nodes which are updated in parallel by using Graphics Processing Unit (GPUs) devices and CUDA programming. By leveraging the sparsity of biological networks and of the available annotated proteins characterizing the AFP context, Par-COSNet adopts a sparse representation for both network connections and protein functions/labels. This, together with the parallel design and the usage of GPU devices, allows to significantly speed-up the computation with ordinary single-species biological networks, and opens the avenue to efficiently predict protein functions in large multi-species networks on ordinary computers, as shown in the experiments performed with synthetic networks having millions of nodes and hundreds of millions of edges.

As a possible future work for the improvement of Par-COSNet, we also have proposed a new parallel algorithm for graph coloring problem based on Markov Chain Monte Carlo techniques. The main goal of this new method is to produce balanced solutions, which is a direction not much explored yet in the literature. Experiments show the effectiveness of the approach on random graphs and real-world graphs. We have studied the convergence properties of the algorithm and we also have sketched a qualitative analysis

of the achieved balancing, remarking that it is as important as difficult to formally derive an analytic form for the distribution on the final colorings, due to the complex dynamics among the service probability distributions the model puts into play during its evolution.

Chapter 6

Conclusions

In this thesis we presented three High Performance Computing Machine Learning solutions specifically designed to tackle current open issues in the research fields of Precision and Genomic Medicine.

We presented *parSMURF*, a novel HPC Machine Learning method for the prediction of deleterious and pathogenic variants in the non-coding area of the genome. Thanks to its fast parallel implementation, capable of scaling on multi-core and multi-node architectures, and its hyper-parameters auto-tuning capability provided by Bayesian Optimization, *parSMURF* represents the current state of the art solution when facing classification problems characterized by highly imbalanced datasets.

We also presented a GPU accelerated Deep Neural Network model for the assessment of the regulatory region type and activity across different cell lines. Our model is in turn composed by two Deep Neural Network classifiers - a Feed-Forward Neural Network trained with epigenomic data and a Convolutional Neural Network trained with sequence data - that are able to discriminate whether a regulatory region is active or not in a given cell line. We also showed the importance of a correct balancing strategy among classes, as its improper use may lead to over-optimistic results, and a novel parallelization method for Bayesian Optimization that sensibly increase the performance of the model without incurring in computational overhead that may limit the exploration of the hyper-parameters space.

Finally, we presented *Par-COSNet*, a Hopfield network -based GPU accelerated method for the solving the labeling problem on partially labeled graphs. Its novel CPU-GPU co-operative parallelization scheme is able to sensibly speed-up the overall computation and, at the same time, keeping the Hopfield dynamics sequential. Also, its efficient memory allocation scheme opens *Par-COSNet* to a whole new range of Big Data applications.

To face some of the open issues of our works, we proposed several on going

and future extensions. We developed a more scalable version of *parSMURF*, called *parSMURF-NG* (New Generation), that will we applied in the project *ParBigMen* (ParSMURF application to Big genomic and epigenomic data for the detection of pathogenic variants in Mendelian diseases). In this European Union funded project, which is recently been awarded 50 Millions core hours in the Liebniz Supercomputing Center by the PRACE consortium, we will evaluate new genome-wide pathogenicity scores to be included in the new version of *Genomizer*.

We are also improving the model presented for the prediction of regulatory regions activity: by observing that epigenomic and sequence data are able to extract different kind of information from the data, we are now currently designing and evaluating a multi-modal DNN model that will improve the prediction capability over the current model.

Finally, we noticed that the computing performance of *Par-COSNet* could further be improved. For this reason, we designed and developed a new parallel probabilistic coloring solution. This strategy is going to be included in the next release of *Par-COSNet*, and provides a balanced workload to the GPU for achieving better performance of the overall execution.

Bibliography

[1] Collins FS. Medical and Societal Consequences of the Human Genome Project. New England Journal of Medicine 1999;341(1):28–37. https://doi.org/10.1056/NEJM199907013410106, pMID: 10387940.

[2] consortium TPMI, of Health U S National Library of Medicine TNI, editor, What is the Precision Medicine Initiative? The National Institute of Health; 2015. https://ghr.nlm.nih.gov/primer/precisionmedicine/initiative, accessed 20/08/2020.

[3] Moscoso CG, Potz KR, Tan S, Jacobson PA, Berger KM, Steer CJ. Precision medicine, agriculture, and genome editing: science and ethics. Annals of the New York Academy of Sciences 2020;1465(1):59–75. https://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/nyas.14266.

[4] König IR, Fuchs O, Hansen G, von Mutius E, Kopp MV. What is precision medicine? European Respiratory Journal 2017;50(4). https://erj.ersjournals.com/content/50/4/1700391.

[5] Pacanowski M, Liu Q. Precision Medicine 2030. Clinical Pharmacology & Therapeutics 2020;107(1):62–64. https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/cpt.1675.

[6] Council NR. Toward Precision Medicine: Building a Knowledge Network for Biomedical Research and a New Taxonomy of Disease. Washington, DC: The National Academies Press; 2011. https://www.nap.edu/catalog/13284/toward-precision-medicine-building-a-knowledge-network-for-biomedical-resea

[7] Goetz LH, Schork NJ. Personalized medicine: motivation, challenges, and progress. Fertility and sterility 2018 Jun;109(6):952–963. https://pubmed.ncbi.nlm.nih.gov/29935653.

[8] McGrath S, Ghersi D. Building towards precision medicine: empowering medical professionals for the next revolution. BMC Medical Genomics 2016 May;9(1):23. https://doi.org/10.1186/s12920-016-0183-8.

[9] Khoury MJ, for Disease Control C, Prevention, editors, The Shift From Personalized Medicine to Precision Medicine and Precision Public Health: Words Matter! U.S. Department of Health and Human Services; 2016. https://blogs.cdc.gov/genomics/2016/04/21/shift/, accessed 20/08/2020.

[10] Kalra S, Das AK, Bajaj S, Priya G, Ghosh S, Mehrotra RN, et al. Utility of Precision Medicine in the Management of Diabetes: Expert Opinion from an International Panel. Diabetes therapy : research, treatment and education of diabetes and related disorders 2020 Feb;11(2):411–422. https://pubmed.ncbi.nlm.nih.gov/31916214.

[11] Ginsburg GS, Phillips KA. Precision Medicine: From Science To Value. Health Affairs 2018;37(5):694–701. https://doi.org/10.1377/hlthaff.2017.1624, pMID: 29733705.

[12] Snyderman R, Meade C, Drake C. Letter to Editor. Journal of American Medical Association 2016 Feb;315(6):613.

[13] Chitnis T, Prat A. A roadmap to precision medicine for multiple sclerosis. Multiple Sclerosis Journal 2020;26(5):522–532. https://doi.org/10.1177/1352458519881558, pMID: 31965904.

[14] Naylor S. What's in a Name? The Evolution of "P-Medicine". Journal of Precision Medicine 2015 Nov;11(11):15–29. http://www.thejournalofprecisionmedicine.com/wp-content/uploads/2015/11/NAYLOR.pdf.

[15] Inc I, Inc I, editor, Illumina: NGS vs. Sanger. Illumina Inc.; 2020. https://www.illumina.com/science/technology/next-generation-sequencing/ngs-vs-sanger-sequencing.html.

[16] Wetterstrand KA, Institute TNHGR, editor, The Cost of Sequencing a Human Genome. The National Institute of Health; 2020. https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost.

[17] Consortium TGP. A global reference for human genetic variation. Nature 2015;526:68–74.

[18] Olivier M, Asmis R, Hawkins GA, Howard TD, Cox LA. The Need for Multi-Omics Biomarker Signatures in Precision Medicine. International journal of molecular sciences 2019 Sep;20(19):4781. https://pubmed.ncbi.nlm.nih.gov/31561483.

[19] McKinney SM, Sieniek M, Godbole V, Godwin J, Antropova N, Ashrafian H, et al. International evaluation of an AI system for breast cancer screening. Nature 2020 Jan;577(7788):89–94. https://doi.org/10.1038/s41586-019-1799-6.

[20] Deiner MS, Lietman TM, McLeod SD, Chodosh J, Porco TC. Surveillance Tools Emerging From Search Engines and Social Media Data for Determining Eye Disease Patterns. JAMA ophthalmology 2016 Sep;134(9):1024–1030. https://pubmed.ncbi.nlm.nih.gov/27416554.

[21] Benke K, Benke G. Artificial Intelligence and Big Data in Public Health. International journal of environmental research and public health 2018 Dec;15(12):2796. https://pubmed.ncbi.nlm.nih.gov/30544648.

[22] Noorbakhsh-Sabet N, Zand R, Zhang Y, Abedi V. Artificial Intelligence Transforms the Future of Health Care. The American journal of medicine 2019 Jul;132(7):795–801. https://pubmed.ncbi.nlm.nih.gov/30710543.

[23] Chang S, Pierson E, Koh PW, Gerardin J, Redbird B, Grusky D, et al. Mobility network models of COVID-19 explain inequities and inform reopening. Nature 2020 Nov;https://doi.org/10.1038/s41586-020-2923-3.

[24] Foundation AS, Foundation AS, editor, Hadoop. Apache Software Foundation; 2010. https://hadoop.apache.org.

[25] Consortium TEP. An integrated encyclopedia of DNA elements in the human genome. Nature 2012 Sep;489(7414):57–74. https://doi.org/10.1038/nature11247.

[26] Noguchi S, Arakawa T, Fukuda S, Furuno M, Hasegawa A, Hori F, et al. FANTOM5 CAGE profiles of human and mouse samples. Scientific Data 2017 Aug;4(1):170112. https://doi.org/10.1038/sdata.2017.112.

[27] Bujold D, Morais DAdL, Gauthier C, Côé C, Caron M, Kwan T, et al. The International Human Epigenome Consortium Data Portal. Cell Systems 2016 Nov;3(5):496–499.e2. https://doi.org/10.1016/j.cels.2016.10.019.

[28] Alexandrov LB, Nik-Zainal S, Wedge DC, Campbell PJ, Stratton MR. Deciphering signatures of mutational processes operative in human cancer. Cell reports 2013 Jan;3(1):246–259. https://pubmed.ncbi.nlm.nih.gov/23318258.

[29] Russell S, Norvig P. Artificial Intelligence: A Modern Approach. Pearson; 2020. http://aima.cs.berkeley.edu/.

[30] Murphy KP. Machine Learning: A Probabilistic Perspective. MIT Press; 2012.

[31] Koprowski R, Foster KR. Machine learning and medicine: book review and commentary. Biomedical engineering online 2018 Feb;17(1):17–17. https://pubmed.ncbi.nlm.nih.gov/29391026.

[32] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag; 2009.

[33] Deo RC. Machine Learning in Medicine. Circulation 2015 Nov;132(20):1920–1930. https://pubmed.ncbi.nlm.nih.gov/26572668.

[34] Campbell C. 12. In: Kasabov N, editor. Machine Learning Methodology in Bioinformatics Berlin, Heidelberg: Springer Berlin Heidelberg; 2014. p. 185–206. https://doi.org/10.1007/978-3-642-30574-0_12.

[35] Zheng A, Casari A. Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists. O'Reilly Media; 2018.

[36] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016. http://www.deeplearningbook.org.

[37] Shahinfar S, Meek P, Falzon G. "How many images do I need?" Understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring. Ecological Informatics 2020;57:101085. http://www.sciencedirect.com/science/article/pii/S1574954120300352.

[38] Chollet F, et al., GitHub, editor, Keras. GitHub; 2015. `https://keras.io`.

[39] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. TensorFlow: A System for Large-Scale Machine Learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation OSDI'16, USA: USENIX Association; 2016. p. 265–283.

[40] Sato K, Platform GC, editor, How a Japanese cucumber farmer is using deep learning and TensorFlow. Google; 2016. `https://cloud.google.com/blog/products/gcp/how-a-japanese-cucumber-farmer-is-using-deep-learning-and-tensorflow`.

[41] Almuhayar M, Lu HH, Iriawan N. Classification of Abnormality in Chest X-Ray Images by Transfer Learning of CheXNet. In: 2019 3rd International Conference on Informatics and Computational Sciences (ICICoS) 2019 3rd International Conference on Informatics and Computational Sciences (ICICoS); 2019. p. 1–6.

[42] Liu J, Chen Y, Lan L, Lin B, Chen W, Wang M, et al. Prediction of rupture risk in anterior communicating artery aneurysms with a feed-forward artificial neural network. European Radiology 2018 Aug;28(8):3268–3275. `https://doi.org/10.1007/s00330-017-5300-3`.

[43] Guillame-Bert M, Dubrawski A, Wang D, Hravnak M, Clermont G, Pinsky MR. Learning temporal rules to forecast instability in continuously monitored patients. Journal of the American Medical Informatics Association : JAMIA 2017 Jan;24(1):47–53. `https://pubmed.ncbi.nlm.nih.gov/27274020`.

[44] Hannun AY, Rajpurkar P, Haghpanahi M, Tison GH, Bourn C, Turakhia MP, et al. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nature Medicine 2019 Jan;25(1):65–69. `https://doi.org/10.1038/s41591-018-0268-3`.

[45] Lu P, Abedi V, Mei Y, Hontecillas R, Hoops S, Carbo A, et al. Supervised learning methods in modeling of CD4+ T cell heterogeneity. BioData mining 2015 Sep;8:27–27. `https://pubmed.ncbi.nlm.nih.gov/26339293`.

[46] Xu J, Yang P, Xue S, Sharma B, Sanchez-Martin M, Wang F, et al. Translating cancer genomics into precision medicine with artificial intelligence: applications, challenges and future perspectives. Human genetics 2019 Feb;138(2):109–124. https://pubmed.ncbi.nlm.nih.gov/30671672.

[47] Song C, Kong Y, Huang L, Luo H, Zhu X. Big data-driven precision medicine: Starting the custom-made era of iatrology. Biomedicine and Pharmacotherapy 2020;129:110445. http://www.sciencedirect.com/science/article/pii/S0753332220306387.

[48] Evans WE, Pui CH, Yang JJ. The Promise and the Reality of Genomics to Guide Precision Medicine in Pediatric Oncology: The Decade Ahead. Clinical Pharmacology & Therapeutics 2020;107(1):176–180. https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/cpt.1660.

[49] Zou J, Wang E. Cancer Biomarker Discovery for Precision Medicine: New Progress. Current Medicinal Chemistry 2019 01;26(42):7655–7671. https://www.ingentaconnect.com/content/ben/cmc/2019/00000026/00000042/art00009.

[50] Guyon I, Weston J, Barnhill S, Vapnik V. Gene Selection for Cancer Classification using Support Vector Machines. Machine Learning 2002 Jan;46(1):389–422. https://doi.org/10.1023/A:1012487302797.

[51] Yousef M, Ketany M, Manevitz L, Showe LC, Showe MK. Classification and biomarker identification using gene network modules and support vector machines. BMC bioinformatics 2009 Oct;10:337–337. https://doi.org/10.1186/1471-2105-10-337.

[52] Mamoshina P, Volosnikova M, Ozerov IV, Putin E, Skibina E, Cortese F, et al. Machine Learning on Human Muscle Transcriptomic Data for Biomarker Discovery and Tissue-Specific Drug Target Identification. Frontiers in genetics 2018 Jul;9:242–242. https://doi.org/10.3389/fgene.2018.00242.

[53] Polikowsky HG, Drake KA. Supervised Machine Learning with CITRUS for Single Cell Biomarker Discovery. Methods in molecular biology (Clifton, NJ) 2019;1989:309–332. https://doi.org/10.1007/978-1-4939-9454-0_20.

[54] Qi B, Fiori LM, Turecki G, Trakadis YJ. Machine Learning Analysis of Blood microRNA Data in Major Depression: A Case-Control Study

for Biomarker Discovery. International Journal of Neuropsychopharmacology 2020 05;https://doi.org/10.1093/ijnp/pyaa029.

[55] Institute NNC, Institute TNC, editor, How Imatinib Transformed Leukemia Treatment and Cancer Research. The National Institute of Health; 2018. https://www.cancer.gov/research/progress/discovery/gleevec, accessed 20/08/2020.

[56] Dreifus C, Times TNY, editor, Researcher Behind the Drug Gleevec. The New York Times; 2009. Accessed 16/02/2020.

[57] Daly AK. Pharmacogenetics: a general review on progress to date. British Medical Bulletin 2017 10;124(1):65–79. https://doi.org/10.1093/bmb/ldx035.

[58] Wheeler HE, Maitland ML, Dolan ME, Cox NJ, Ratain MJ. Cancer pharmacogenomics: strategies and challenges. Nature Reviews Genetics 2013 Jan;14(1):23–34. https://doi.org/10.1038/nrg3352.

[59] Relling MV, Evans WE. Pharmacogenomics in the clinic. Nature 2015 Oct;526(7573):343–350. https://doi.org/10.1038/nature15817.

[60] Fleming N. Computer-calculated compounds. Nature 2018 May;557(1):55–58. https://www.nature.com/articles/d41586-018-05267-x.

[61] Pushpakom S, Iorio F, Eyers PA, Escott KJ, Hopper S, Wells A, et al. Drug repurposing: progress, challenges and recommendations. Nature Reviews Drug Discovery 2019 Jan;18(1):41–58. https://doi.org/10.1038/nrd.2018.168.

[62] Fernandez-Granero M, Sanchez-Morillo D, Leon-Jimenez A. Computerised Analysis of Telemonitored Respiratory Sounds for Predicting Acute Exacerbations of COPD. Sensors 2015 Oct;15(10):26978–26996. https://doi.org/10.3390/s151026978.

[63] Kusmakar S, Karmakar CK, Yan B, O'Brien TJ, Muthuganapathy R, Palaniswami M. Detection of generalized tonic-clonic seizures using short length accelerometry signal. In: 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC); 2017. p. 4566–4569. https://doi.org/10.1109/EMBC.2017.8037872.

[64] Sverdlov O, van Dam J, Hannesdottir K, Thornton-Wells T. Digital Therapeutics: An Integral Component of Digital Innovation in Drug Development. Clinical Pharmacology & Therapeutics 2018;104(1):72–80. https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/cpt.1036.

[65] Desai AN. Artificial Intelligence: Promise, Pitfalls, and Perspective. JAMA 2020 06;323(24):2448–2449. https://doi.org/10.1001/jama.2020.8737.

[66] Ohler U, Liao Gc, Niemann H, Rubin GM. Computational analysis of core promoters in the Drosophila genome. Genome biology 2002;3(12):RESEARCH0087–RESEARCH0087. https://pubmed.ncbi.nlm.nih.gov/12537576.

[67] Heintzman ND, Stuart RK, Hon G, Fu Y, Ching CW, Hawkins RD, et al. Distinct and predictive chromatin signatures of transcriptional promoters and enhancers in the human genome. Nature Genetics 2007 Mar;39(3):311–318. https://doi.org/10.1038/ng1966.

[68] Consortium IHGS. Initial sequencing and analysis of the human genome. Nature 2001 Feb;409(6822):860–921. https://doi.org/10.1038/35057062.

[69] Claussnitzer M, Cho JH, Collins R, Cox NJ, Dermitzakis ET, Hurles ME, et al. A brief history of human disease genetics. Nature 2020 Jan;577(7789):179–189. https://doi.org/10.1038/s41586-019-1879-7.

[70] Kremer B, Goldberg P, Andrew SE, Theilmann J, Telenius H, Zeisler J, et al. A Worldwide Study of the Huntington's Disease Mutation: The Sensitivity and Specificity of Measuring CAG Repeats. New England Journal of Medicine 1994;330(20):1401–1406. https://doi.org/10.1056/NEJM199405193302001, pMID: 8159192.

[71] Gusella JF, MacDonald ME. Huntington's Disease and Repeating Trinucleotides. New England Journal of Medicine 1994;330(20):1450–1451. https://doi.org/10.1056/NEJM199405193302011, pMID: 8159202.

[72] Gregory TR, editor. The Evolution of the Genome. Burlington: Academic Press; 2005. http://www.sciencedirect.com/science/article/pii/B9780123014634500000.

[73] Nakagawa H, Fujita M. Whole genome sequencing analysis for cancer genomics and precision medicine. Cancer Science 2018;109(3):513–522.

[74] Adams DR, Eng CM. Next-Generation Sequencing to Diagnose Suspected Genetic Disorders. N Engl J Med 2018;379:1353–62.

[75] Visscher PM, Wray NR, Zhang Q, Sklar P, McCarthy MI, Brown MA, et al. 10 Years of GWAS Discovery: Biology, Function, and Translation. The American Journal of Human Genetics 2017 Jul;101(1):5–22. https://doi.org/10.1016/j.ajhg.2017.06.005.

[76] Committee PGCC, Cichon S, Craddock N, Daly M, Faraone SV, Gejman PV, et al. Genomewide association studies: history, rationale, and prospects for psychiatric disorders. The American journal of psychiatry 2009 May;166(5):540–556. https://pubmed.ncbi.nlm.nih.gov/19339359.

[77] Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. American journal of human genetics 2007 Sep;81(3):559–575. https://pubmed.ncbi.nlm.nih.gov/17701901.

[78] Aulchenko YS, Struchalin MV, van Duijn CM. ProbABEL package for genome-wide association analysis of imputed data. BMC Bioinformatics 2010 Mar;11(1):134. https://doi.org/10.1186/1471-2105-11-134.

[79] Pei YF, Zhang L, Li J, Deng HW. Analyses and comparison of imputation-based association methods. PloS one 2010 May;5(5):e10827–e10827. https://pubmed.ncbi.nlm.nih.gov/20520814.

[80] Ozaki K, Ohnishi Y, Iida A, Sekine A, Yamada R, Tsunoda T, et al. Functional SNPs in the lymphotoxin-α gene that are associated with susceptibility to myocardial infarction. Nature Genetics 2002 Dec;32(4):650–654. https://doi.org/10.1038/ng1047.

[81] Jones CC, Bush WS, Crawford DC, Wenzlaff AS, Schwartz AG, Wiencke JK, et al. Germline Genetic Variants and Lung Cancer Survival in African Americans. Cancer epidemiology, biomarkers & prevention : a publication of the American Association for Cancer

Research, cosponsored by the American Society of Preventive Oncology 2017 Aug;26(8):1288–1295. https://pubmed.ncbi.nlm.nih.gov/28619829.

[82] Chat V, Ferguson R, Simpson D, Kazlow E, Lax R, Moran U, et al. Autoimmune genetic variants as germline biomarkers of response in melanoma immunotherapy treatment. Journal of Clinical Oncology 2018;36(15_suppl):3079–3079. https://doi.org/10.1200/JCO.2018.36.15_suppl.3079.

[83] Lambert JC, Ibrahim-Verbaas CA, Harold D, Naj AC, Sims R, Bellenguez C, et al. Meta-analysis of 74,046 individuals identifies 11 new susceptibility loci for Alzheimer's disease. Nature genetics 2013 Dec;45(12):1452–1458. https://pubmed.ncbi.nlm.nih.gov/24162737.

[84] Mills MC, Rahal C. A scientometric review of genome-wide association studies. Communications Biology 2019 Jan;2(1):9. https://doi.org/10.1038/s42003-018-0261-x.

[85] Tam V, Patel N, Turcotte M, Bossé Y, Paré G, Meyre D. Benefits and limitations of genome-wide association studies. Nature Reviews Genetics 2019 Aug;20(8):467–484. https://doi.org/10.1038/s41576-019-0127-1.

[86] Ng PC, Henikoff S. Predicting deleterious amino acid substitutions. Genome research 2001 May;11(5):863–874. https://pubmed.ncbi.nlm.nih.gov/11337480.

[87] Thomas PD, Campbell MJ, Kejariwal A, Mi H, Karlak B, Daverman R, et al. PANTHER: a library of protein families and subfamilies indexed by function. Genome research 2003 Sep;13(9):2129–2141. https://pubmed.ncbi.nlm.nih.gov/12952881.

[88] Capriotti E, Calabrese R, Casadio R. Predicting the insurgence of human genetic diseases associated to single point protein mutations with support vector machines and evolutionary information. Bioinformatics 2006 08;22(22):2729–2734. https://doi.org/10.1093/bioinformatics/btl423.

[89] Bao L, Zhou M, Cui Y. nsSNPAnalyzer: identifying disease-associated nonsynonymous single nucleotide polymorphisms. Nucleic acids research 2005 Jul;33(Web Server issue):W480–W482. https://pubmed.ncbi.nlm.nih.gov/15980516.

[90] Bromberg Y, Rost B. SNAP: predict effect of non-synonymous polymorphisms on function. Nucleic acids research 2007;35(11):3823–3835. https://pubmed.ncbi.nlm.nih.gov/17526529.

[91] Li B, Krishnan VG, Mort ME, Xin F, Kamati KK, Cooper DN, et al. Automated inference of molecular mechanisms of disease from amino acid substitutions. Bioinformatics (Oxford, England) 2009 Nov;25(21):2744–2750. https://pubmed.ncbi.nlm.nih.gov/19734154.

[92] Adzhubei IA, Schmidt S, Peshkin L, Ramensky VE, Gerasimova A, Bork P, et al. A method and server for predicting damaging missense mutations. Nature methods 2010 Apr;7(4):248–249. https://pubmed.ncbi.nlm.nih.gov/20354512.

[93] Calabrese R, Capriotti E, Fariselli P, Martelli PL, Casadio R. Functional annotations improve the predictive score of human disease-related mutations in proteins. Human Mutation 2009;30(8):1237–1244. https://onlinelibrary.wiley.com/doi/abs/10.1002/humu.21047.

[94] Ioannidis NM, Rothstein JH, Pejaver V, Middha S, McDonnell SK, Baheti S, et al. REVEL: An Ensemble Method for Predicting the Pathogenicity of Rare Missense Variants. The American Journal of Human Genetics 2016;99(4):877 – 885. http://www.sciencedirect.com/science/article/pii/S0002929716303706.

[95] Chennen K, Weber T, Lornage X, Kress A, Böhm J, Thompson J, et al. MISTIC: A prediction tool to reveal disease-relevant deleterious missense variants. PloS one 2020 Jul;15(7):e0236962–e0236962. https://pubmed.ncbi.nlm.nih.gov/32735577.

[96] Alirezaie N, Kernohan KD, Hartley T, Majewski J, Hocking TD. ClinPred: Prediction Tool to Identify Disease-Relevant Nonsynonymous Single-Nucleotide Variants. American journal of human genetics 2018 Oct;103(4):474–483. https://pubmed.ncbi.nlm.nih.gov/30220433.

[97] Sundaram L, Gao H, Padigepati SR, McRae JF, Li Y, Kosmicki JA, et al. Predicting the clinical impact of human mutation with deep neural networks. Nature genetics 2018 Aug;50(8):1161–1170. https://pubmed.ncbi.nlm.nih.gov/30038395.

[98] Kircher M, Witten DM, Jain P, O'Roak BJ, Cooper GM, Shendure J. A general framework for estimating the relative pathogenicity of human genetic variants. Nat Genet 2014 Mar;46(3):310–315.

[99] Rentzsch P, Witten D, Cooper G, Shendure J, Kircher M. CADD: predicting the deleteriousness of variants throughout the human genome. Nucleic Acids Res 2019;47(D1):D886–D894.

[100] Quang D, Xie X, Chen Y. DANN: a deep learning approach for annotating the pathogenicity of genetic variants. Bioinformatics 2014 10;31(5):761–763. `https://dx.doi.org/10.1093/bioinformatics/btu703`.

[101] Ritchie GRS, Dunham I, Zeggini E, Flicek P. Functional annotation of noncoding sequence variants. Nat Methods 2014 Mar;11(3):294–296. `http://dx.doi.org/10.1038/nmeth.2832`.

[102] Shihab HA, Rogers MF, Gough J, Mort M, Cooper DN, Day IN, et al. An integrative approach to predicting the functional effects of non-coding and coding sequence variation. Bioinformatics 2015;31(10):1536–1543.

[103] Huang YF, Gulko B, Siepel A. Fast, scalable prediction of deleterious noncoding variants from functional and population genomic data. Nature Genetics 2017;49:618–624.

[104] Ionita-Laza I, McCallum K, Xu B, Buxbaum JD. A spectral approach integrating functional genomic annotations for coding and noncoding variants. Nat Genet 2016 Feb;48(2):214–20.

[105] Zhou J, Troyanskaya OG. Predicting effects of noncoding variants with deep learning-based sequence model. Nature Methods 2015 Aug;12(10):931–934. `http://dx.doi.org/10.1038/nmeth.3547`.

[106] Alipanahi B, Delong A, Weirauch MT, Frey BJ. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nature Biotechnology 2015 Aug;33(8):831–838. `https://doi.org/10.1038/nbt.3300`.

[107] Lee D, Gorkin DU, Baker M, Strober BJ, Asoni AL, McCallion AS, et al. A method to predict the impact of regulatory variants from DNA sequence. Nature genetics 2015;47(8):955.

[108] Smedley D, Schubach M, Jacobsen JOB, Köhler S, Zemojtel T, Spielmann M, et al. A Whole-Genome Analysis Framework for Effective Identification of Pathogenic Regulatory Variants in Mendelian Disease. The American Journal of Human Genetics 2016 sep;99(3):595–606. http://linkinghub.elsevier.com/retrieve/pii/S0002929716302786.

[109] Schubach M, Re M, Robinson PN, Valentini G. Imbalance-Aware Machine Learning for Predicting Rare and Common Disease-Associated Non-Coding Variants. Scientific Reports 2017;7(1):2959.

[110] Caron B, Luo Y, Rausell A. NCBoost classifies pathogenic non-coding variants in Mendelian diseases through supervised learning on purifying selection signals in humans. Genome Biology 2019 Feb;20(1):32. https://doi.org/10.1186/s13059-019-1634-2.

[111] Mora A, Sandve GK, Gabrielsen OS, Eskeland R. In the loop: promoter–enhancer interactions and bioinformatics. Briefings in Bioinformatics 2015 11;17(6):980–995. https://doi.org/10.1093/bib/bbv097.

[112] Smale ST, Kadonaga JT. The RNA Polymerase II Core Promoter. Annual Review of Biochemistry 2003;72(1):449–479. https://doi.org/10.1146/annurev.biochem.72.121801.161520, pMID: 12651739.

[113] Melamed P, Yosefzon Y, Rudnizky S, Pnueli L. Transcriptional enhancers: Transcription, function and flexibility. Transcription 2016;7(1):26–31. https://pubmed.ncbi.nlm.nih.gov/26934309.

[114] Li K, Zhang Y, Liu X, Liu Y, Gu Z, Cao H, et al. Noncoding Variants Connect Enhancer Dysregulation with Nuclear Receptor Signaling in Hematopoietic Malignancies. Cancer discovery 2020 May;10(5):724–745. https://pubmed.ncbi.nlm.nih.gov/32188707.

[115] McClymont SA, Hook PW, Soto AI, Reed X, Law WD, Kerans SJ, et al. Parkinson-Associated SNCA Enhancer Variants Revealed by Open Chromatin in Mouse Dopamine Neurons. American journal of human genetics 2018 Dec;103(6):874–892. https://doi.org/10.1016/j.ajhg.2018.10.018.

[116] Dudek AM, Vermeulen SH, Kolev D, Grotenhuis AJ, Kiemeney LALM, Verhaegh GW. Identification of an enhancer region within the TP63/LEPREL1 locus containing genetic variants associated with

bladder cancer risk. Cellular oncology (Dordrecht) 2018 Oct;41(5):555–568. https://doi.org/10.1007/s13402-018-0393-5.

[117] Corradin O, Scacheri PC. Enhancer variants: evaluating functions in common disease. Genome medicine 2014 Oct;6(10):85–85. https://doi.org/10.1186/s13073-014-0085-3.

[118] Gao T, Qian J. EAGLE: An algorithm that utilizes a small number of genomic features to predict tissue/cell type-specific enhancer-gene interactions. PLoS computational biology 2019 Oct;15(10):e1007436–e1007436. https://doi.org/10.1371/journal.pcbi.1007436.

[119] Hnisz D, Abraham BJ, Lee TI, Lau A, Saint-André V, Sigova AA, et al. Super-enhancers in the control of cell identity and disease. Cell 2013 Nov;155(4):934–947. https://doi.org/10.1016/j.cell.2013.09.053.

[120] Coppola CJ, C Ramaker R, Mendenhall EM. Identification and function of enhancers in the human genome. Human Molecular Genetics 2016 07;25(R2):R190–R197. https://doi.org/10.1093/hmg/ddw216.

[121] Liu Y, Yu S, Dhiman VK, Brunetti T, Eckart H, White KP. Functional assessment of human enhancer activities using whole-genome STARR-sequencing. Genome Biology 2017 Nov;18(1):219. https://doi.org/10.1186/s13059-017-1345-5.

[122] Ernst J, Kellis M. ChromHMM: automating chromatin-state discovery and characterization. Nature Methods 2012 Mar;9(3):215–216. https://doi.org/10.1038/nmeth.1906.

[123] Hoffman MM, Buske OJ, Wang J, Weng Z, Bilmes JA, Noble WS. Unsupervised pattern discovery in human chromatin structure through genomic segmentation. Nature Methods 2012 May;9(5):473–476. https://doi.org/10.1038/nmeth.1937.

[124] Yip KY, Cheng C, Bhardwaj N, Brown JB, Leng J, Kundaje A, et al. Classification of human genomic regions based on experimentally determined binding sites of more than 100 transcription-related factors. Genome Biology 2012 Sep;13(9):R48. https://doi.org/10.1186/gb-2012-13-9-r48.

[125] Rajagopal N, Xie W, Li Y, Wagner U, Wang W, Stamatoyannopoulos J, et al. RFECS: a random-forest based algorithm for enhancer identification from chromatin state. PLoS computational biology

2013;9(3):e1002968–e1002968. https://doi.org/10.1371/journal.pcbi.1002968.

[126] Kleftogiannis D, Kalnis P, Bajic VB. DEEP: a general computational framework for predicting enhancers. Nucleic acids research 2015 Jan;43(1):e6–e6. https://pubmed.ncbi.nlm.nih.gov/25378307.

[127] Lu Y, Qu W, Shan G, Zhang C. DELTA: A Distal Enhancer Locating Tool Based on AdaBoost Algorithm and Shape Features of Chromatin Modifications. PloS one 2015 Jun;10(6):e0130622–e0130622. https://pubmed.ncbi.nlm.nih.gov/26091399.

[128] Danko CG, Hyland SL, Core LJ, Martins AL, Waters CT, Lee HW, et al. Identification of active transcriptional regulatory elements from GRO-seq data. Nature methods 2015 May;12(5):433–438. https://doi.org/10.1038/nmeth.3329.

[129] Huang F, Shen J, Guo Q, Shi Y. eRFSVM: a hybrid classifier to predict enhancers-integrating random forests with support vector machines. Hereditas 2016 Jun;153:6–6. https://doi.org/10.1186/s41065-016-0012-2.

[130] van Duijvenboden K, de Boer BA, Capon N, Ruijter JM, Christoffels VM. EMERGE: a flexible modelling framework to predict genomic regulatory elements from genomic signatures. Nucleic acids research 2016 Mar;44(5):e42–e42. https://doi.org/10.1093/nar/gkv1144.

[131] He Y, Gorkin DU, Dickel DE, Nery JR, Castanon RG, Lee AY, et al. Improved regulatory element prediction based on tissue-specific local epigenomic signatures. Proceedings of the National Academy of Sciences of the United States of America 2017 Feb;114(9):E1633–E1640. https://doi.org/10.1073/pnas.1618353114.

[132] Osmala M, Lähdesmäki H. Enhancer prediction in the human genome by probabilistic modelling of the chromatin feature patterns. BMC bioinformatics 2020 Jul;21(1):317–317. https://doi.org/10.1186/s12859-020-03621-3.

[133] Liu F, Li H, Ren C, Bo X, Shu W. PEDLA: predicting enhancers with a deep learning-based algorithmic framework. Scientific Reports 2016 Jun;6(1):28517. https://doi.org/10.1038/srep28517.

[134] Kim SG, Harwani M, Grama A, Chaterji S. EP-DNN: A Deep Neural Network-Based Global Enhancer Prediction Algorithm. Scientific reports 2016 Dec;6:38433–38433. https://doi.org/10.1038/srep38433.

[135] Yang B, Liu F, Ren C, Ouyang Z, Xie Z, Bo X, et al. BiRen: predicting enhancers with a deep-learning-based model using the DNA sequence alone. Bioinformatics 2017 02;33(13):1930–1936. https://doi.org/10.1093/bioinformatics/btx105.

[136] Min X, Zeng W, Chen S, Chen N, Chen T, Jiang R. Predicting enhancers with deep convolutional neural networks. BMC Bioinformatics 2017 Dec;18(13):478. https://doi.org/10.1186/s12859-017-1878-3.

[137] Fang CH, Theera-Ampornpunt N, Roth MA, Grama A, Chaterji S. AIKYATAN: mapping distal regulatory elements using convolutional learning on GPU. BMC bioinformatics 2019 Oct;20(1):488–488. https://doi.org/10.1186/s12859-019-3049-1.

[138] Kelley DR, Snoek J, Rinn JL. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. Genome research 2016 Jul;26(7):990–999. https://doi.org/10.1101/gr.200535.115.

[139] Kelley DR, Reshef YA, Bileschi M, Belanger D, McLean CY, Snoek J. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. Genome research 2018 May;28(5):739–750. https://doi.org/10.1101/gr.227819.117.

[140] Li Y, Shi W, Wasserman WW. Genome-wide prediction of cis-regulatory regions using supervised deep learning methods. BMC Bioinformatics 2018 May;19(1):202. https://doi.org/10.1186/s12859-018-2187-1.

[141] Nair S, Kim DS, Perricone J, Kundaje A. Integrating regulatory DNA sequence and gene expression to predict genome-wide chromatin accessibility across cellular contexts. Bioinformatics (Oxford, England) 2019 Jul;35(14):i108–i116. https://doi.org/10.1093/bioinformatics/btz352.

[142] Movva R, Greenside P, Marinov GK, Nair S, Shrikumar A, Kundaje A. Deciphering regulatory DNA sequences and noncoding genetic variants

using neural network models of massively parallel reporter assays. PloS one 2019 Jun;14(6):e0218073–e0218073. https://doi.org/10.1371/journal.pone.0218073.

[143] Nguyen QH, Nguyen-Vo TH, Le NQK, Do TTT, Rahardja S, Nguyen BP. iEnhancer-ECNN: identifying enhancers and their strength using ensembles of convolutional neural networks. BMC genomics 2019 Dec;20(Suppl 9):951–951. https://doi.org/10.1186/s12864-019-6336-3.

[144] Kopp W, Monti R, Tamburrini A, Ohler U, Akalin A. Deep learning for genomics using Janggu. Nature communications 2020 Jul;11(1):3488–3488. https://doi.org/10.1038/s41467-020-17155-y.

[145] Chen KM, Cofer EM, Zhou J, Troyanskaya OG. Selene: a PyTorch-based deep learning library for sequence data. Nature methods 2019 Apr;16(4):315–318. https://doi.org/10.1038/s41592-019-0360-8.

[146] Lunshof JE, Chadwick R, Vorhaus DB, Church GM. From genetic privacy to open consent. Nature Reviews Genetics 2008 May;9(5):406–411. https://doi.org/10.1038/nrg2360.

[147] Chadwick R, Levitt M, Shickle D. The Right to Know and the Right not to Know. Genetic privacy and responsibility. Cambridge University Press; 2014.

[148] Love-Koh J, Peel A, Rejon-Parrilla JC, Ennis K, Lovett R, Manca A, et al. The Future of Precision Medicine: Potential Impacts for Health Technology Assessment. PharmacoEconomics 2018 Dec;36(12):1439–1451. https://pubmed.ncbi.nlm.nih.gov/30003435.

[149] for Health NI, Excellence C. Evaluation consultation document – Eliglustat for treating type 1 Gaucher disease. National Institute for Health and Care Excellence; 2017. https://www.nice.org.uk/guidance/hst5/documents/evaluation-consultation-document.

[150] Powledge TM, Project GL, editor, That 'Precision Medicine' initiative? A Reality Check. Genetic Literacy Project; 2015. https://geneticliteracyproject.org/2015/02/03/that-precision-medicine-initiative-a-reality-check/.

[151] Joyner J M, Paneth N. Response to Letters. Journal of American Medical Association 2016 Feb;315(6):613–614.

[152] Alawi F. Accounting for diversity in rare disease research and precision medicine. Oral Surgery, Oral Medicine, Oral Pathology and Oral Radiology 2020 Mar;129(3):175–176. https://doi.org/10.1016/j.oooo.2019.11.016.

[153] Price I W Nicholson, Gerke S, Cohen IG. Potential Liability for Physicians Using Artificial Intelligence. JAMA 2019 11;322(18):1765–1766. https://doi.org/10.1001/jama.2019.15064.

[154] Bera K, Schalper KA, Rimm DL, Velcheti V, Madabhushi A. Artificial intelligence in digital pathology - new tools for diagnosis and precision oncology. Nature reviews Clinical oncology 2019 Nov;16(11):703–715. https://pubmed.ncbi.nlm.nih.gov/31399699.

[155] Joyner MJ, Paneth N. Promises, promises, and precision medicine. The Journal of clinical investigation 2019 Mar;129(3):946–948. https://pubmed.ncbi.nlm.nih.gov/30688663.

[156] Gilvary C, Madhukar N, Elkhader J, Elemento O. The Missing Pieces of Artificial Intelligence in Medicine. Trends in Pharmacological Sciences 2019;40(8):555 – 564. http://www.sciencedirect.com/science/article/pii/S0165614719301312, special Issue: Rise of Machines in Medicine.

[157] Stewart J, Sprivulis P, Dwivedi G. Artificial intelligence and machine learning in emergency medicine. Emergency Medicine Australasia 2018;30(6):870–874. https://onlinelibrary.wiley.com/doi/abs/10.1111/1742-6723.13145.

[158] Shaywitz D, Forbes, editor, AI Doesn't Ask Why – But Physicians And Drug Developers Want To Know. Forbes Media LLC; 2018. https://www.forbes.com/sites/davidshaywitz/2018/11/09/ai-doesnt-ask-why-but-physicians-and-drug-developers-want-to-know, accessed 20/08/2020.

[159] Anderson C, OMICs C, editor, AI Influencing Precision Medicine but May Not Match the Hype. Genetic Engineering and Biotechnology News; 2018. https://www.clinicalomics.com/topics/informatics-topic/bioinformatics/ai-influencing-precision-medicine-but-may-not-match-the-hype/, accessed 20/08/2020.

[160] Ibrahim R, Pasic M, Yousef GM. Omics for personalized medicine: defining the current we swim in. Expert Review of Molecular Diagnostics 2016;16(7):719–722. https://doi.org/10.1586/14737159.2016.1164601.

[161] Li MM, Datto M, Duncavage EJ, Kulkarni S, Lindeman NI, Roy S, et al. Standards and Guidelines for the Interpretation and Reporting of Sequence Variants in Cancer: A Joint Consensus Recommendation of the Association for Molecular Pathology, American Society of Clinical Oncology, and College of American Pathologists. The Journal of Molecular Diagnostics 2017;19(1):4 – 23. http://www.sciencedirect.com/science/article/pii/S1525157816302239.

[162] White IR, Royston P, Wood AM. Multiple imputation using chained equations: Issues and guidance for practice. Statistics in Medicine 2011;30(4):377–399. https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.4067.

[163] Abedi V, Shivakumar MK, Lu P, Hontecillas R, Leber A, Ahuja M, et al., PMC E, editor, Latent-Based Imputation of Laboratory Measures from Electronic Health Records: Case for Complex Diseases. bioRxiv; 2018. http://europepmc.org/abstract/PPR/PPR17891.

[164] Saito T, Rehmsmeier M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLoS ONE 2015;10:e0118432.

[165] Shrikumar A, Greenside P, Kundaje A. Learning Important Features Through Propagating Activation Differences. In: Precup D, Teh YW, editors. Proceedings of the 34th International Conference on Machine Learning - Volume 70, vol. 70 of Proceedings of Machine Learning Research International Convention Centre, Sydney, Australia: PMLR; 2017. p. 3145–3153. http://proceedings.mlr.press/v70/shrikumar17a.html.

[166] Survey USG, Survey USG, editor, What Is High-Performance Computing? United States Department of the Interior; 2020. https://www.usgs.gov/core-science-systems/sas/arc/about/what-high-performance-computing, accessed 05/09/20120.

[167] NetApp, NetApp, editor, What Is High-Performance Computing? NetApp; 2020. https://www.netapp.com/us/

info/what-is-high-performance-computing.aspx, accessed 07/09/20120.

[168] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster Computing with Working Sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing HotCloud'10, USA: USENIX Association; 2010. p. 10.

[169] White T. Hadoop: The Definitive Guide. 3rd ed. San Francisco, CA, USA: Yahoo Press; 2012.

[170] Moore GE. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. IEEE Solid-State Circuits Society Newsletter 2006;11(3):33–35. https://doi.org/10.1109/N-SSC.2006.4785860.

[171] Moore GE. Progress in digital integrated electronics [Technical literaiture, Copyright 1975 IEEE. Reprinted, with permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, pp. 11-13.]. IEEE Solid-State Circuits Society Newsletter 2006;11(3):36–37. https://doi.org/10.1109/N-SSC.2006.4804410.

[172] Brock D. 7. In: Brock D, editor. Moore's law at 40 Chemical Heritage Foundation; 2006. p. 122.

[173] Dagum L, Menon R. OpenMP: an industry standard API for shared-memory programming. Computational Science & Engineering, IEEE 1998;5(1):46–55.

[174] Jeffers J, Reinders J. Intel Xeon Phi Coprocessor High Performance Programming 1st Edition. 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2013.

[175] Akeley K. Reality Engine Graphics. In: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH '93, New York, NY, USA: Association for Computing Machinery; 1993. p. 109–116. https://doi.org/10.1145/166117.166131.

[176] Cook S. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs. 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2012.

[177] Company TTSQ, Company TTSQ, editor, TIOBE Index; 2020. https://www.tiobe.com/tiobe-index/, accessed 15 November 2020.

[178] Forum MP, MPI: A Message-Passing Interface Standard. Knoxville, TN, USA: University of Tennessee; 1994. https://www.mpi-forum.org/, accessed 07/15/2018.

[179] Project TOM, Project TOM, editor, Open MPI: Open Source High Performance Computing. The Open MPI Project; 2004. https://www.open-mpi.org/, accessed 20/09/2020.

[180] Corporation I, Corporation I, editor, Intel MPI Library. Intel Corporation; 2004. https://software.intel.com/content/www/us/en/develop/tools/mpi-library.html, accessed 20/09/2020.

[181] Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K. Apache Flink™: Stream and Batch Processing in a Single Engine. IEEE Data Eng Bull 2015;38(4):28–38. http://sites.computer.org/debull/A15dec/p28.pdf.

[182] Saha B, Shah H, Seth S, Vijayaraghavan G, Murthy A, Curino C. Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data SIGMOD '15, New York, NY, USA: Association for Computing Machinery; 2015. p. 1357–1369. https://doi.org/10.1145/2723372.2742790.

[183] Akidau T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, et al. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, out-of-Order Data Processing. Proc VLDB Endow 2015 Aug;8(12):1792–1803. https://doi.org/10.14778/2824032.2824076.

[184] Kang SJ, Lee SY, Lee KM. Performance comparison of OpenMP, MPI, and MapReduce in practical problems. Advances in Multimedia 2015;2015:7.

[185] Corporation M, Corporation M, editor, Microsoft Cognitive Toolkit - Official GirHub repository; 2019. https://github.com/Microsoft/CNTK.

[186] Costa FF. Big data in biomedicine. Drug Discovery Today 2014;19(4):433 – 440. http://www.sciencedirect.com/science/article/pii/S1359644613003644.

[187] Grobe H, Diepenbroek M, Dittert N, Reinke M, Sieger R. In: Fütterer DK, Damaske D, Kleinschmidt G, Miller H, Tessensohn F, editors. Archiving and Distributing Earth-Science Data with the PANGAEA Information System Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. p. 403–406. https://doi.org/10.1007/3-540-32934-X_51.

[188] Merelli I, Pérez-Sánchez H, Gesing S, D'Agostino D. Managing, analysing, and integrating big data in medical bioinformatics: open problems and future perspectives. BioMed research international 2014;2014:134023–134023. https://doi.org/10.1155/2014/134023.

[189] Liu T, Lu D, Zhang H, Zheng M, Yang H, Xu Y, et al. Applying high-performance computing in drug discovery and molecular simulation. National science review 2016 Mar;3(1):49–63. https://doi.org/10.1093/nsr/nww003.

[190] Peng S, Zhang X, Lu Y, Liao X, Lu K, Yang C, et al. mAMBER: A CPU/MIC collaborated parallel framework for AMBER on Tianhe-2 supercomputer. In: 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM); 2016. p. 651–657. https://doi.org/10.1109/BIBM.2016.7822595.

[191] Peng S, Cui Y, Yang S, Su W, Zhang X, Zhang T, et al. A CPU/MIC Collaborated Parallel Framework for GROMACS on Tianhe-2 Supercomputer. IEEE/ACM Transactions on Computational Biology and Bioinformatics 2019;16(2):425–433.

[192] Vázquez M, Arís R, Houzeaux G, Aubry R, Villar P, Garcia-Barnés J, et al. A massively parallel computational electrophysiology model of the heart. International Journal for Numerical Methods in Biomedical Engineering;27(12):1911–1929. https://doi.org/10.1002/cnm.1443.

[193] Bordas R, Carpentieri B, Fotia G, Maggio F, Nobes R, Pitt-Francis J, et al. Simulation of cardiac electrophysiology on next-generation high-performance computers. Philosophical transactions Series A, Mathematical, physical, and engineering sciences 2009 06;367:1951–69.

[194] Nageswaran JM, Dutt N, Krichmar JL, Nicolau A, Veidenbaum AV. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. Neural Networks 2009;22(5):791 – 800. http://www.sciencedirect.com/

science/article/pii/S0893608009001373, advances in Neural Networks Research: IJCNN2009.

[195] Neofytou A, Chatzikonstantis G, Magkanaris I, Smaragdos G, Strydis C, Soudris D. GPU Implementation of Neural-Network Simulations Based on Adaptive-Exponential Models. In: 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE) 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE); 2019. p. 339–343. https://doi.org/10.1109/BIBE.2019.00067.

[196] Zhou Y, Liepe J, Sheng X, Stumpf MPH, Barnes C. GPU accelerated biochemical network simulation. Bioinformatics (Oxford, England) 2011 Mar;27(6):874–876. https://doi.org/10.1093/bioinformatics/btr015.

[197] Zhu Z, Tong X, Zhu Z, Liang M, Cui W, Su K, et al. Development of GMDR-GPU for gene-gene interaction analysis and its application to WTCCC GWAS data for type 2 diabetes. PloS one 2013 Apr;8(4):e61943–e61943. https://doi.org/10.1371/journal.pone.0061943.

[198] Ueki M, Tamiya G. Ultrahigh-dimensional variable selection method for whole-genome gene-gene interaction analysis. BMC bioinformatics 2012 May;13:72–72. https://doi.org/10.1186/1471-2105-13-72.

[199] Kam-Thong T, Czamara D, Tsuda K, Borgwardt K, Lewis CM, Erhardt-Lehmann A, et al. EPIBLASTER-fast exhaustive two-locus epistasis detection strategy using graphical processing units. European journal of human genetics : EJHG 2011 Apr;19(4):465–471. https://doi.org/10.1038/ejhg.2010.196.

[200] Kam-Thong T, Azencott CA, Cayton L, Pütz B, Altmann A, Karbalai N, et al. GLIDE: GPU-Based Linear Regression for Detection of Epistasis. Human Heredity 2012;73(4):220–236. https://doi.org/10.1159/000341885.

[201] Wang X, Wu C, Huang Z. Trends of Intel MIC Application In Bioinformatics. In: 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS) 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS); 2019. p. 638–641. https://doi.org/10.1109/ICSESS47205.2019.9040682.

[202] Kozlov AM, Goll C, Stamatakis A. Efficient Computation of the Phylogenetic Likelihood Function on the Intel MIC Architecture. In: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops 2014 IEEE International Parallel & Distributed Processing Symposium Workshops; 2014. p. 518–527. https://doi.org/10.1109/IPDPSW.2014.198.

[203] Kozlov AM, Aberer AJ, Stamatakis A. ExaML version 3: a tool for phylogenomic analyses on supercomputers. Bioinformatics (Oxford, England) 2015 Aug;31(15):2577–2579. https://doi.org/10.1093/bioinformatics/btv184.

[204] Peng S, Yang S, Gao M, Liao X, Liu J, Yang C, et al. P-Hint-Hunt: a deep parallelized whole genome DNA methylation detection tool. BMC genomics 2017 Mar;18(Suppl 2):134–134. https://doi.org/10.1186/s12864-017-3497-9.

[205] Khan MA, Elias I, Sjölund E, Nylander K, Guimera RV, Schobesberger R, et al. Fastphylo: fast tools for phylogenetics. BMC bioinformatics 2013 Nov;14:334–334. https://doi.org/10.1186/1471-2105-14-334.

[206] Kim T, Joo H. ClustalXeed: a GUI-based grid computation version for high performance and terabyte size multiple sequence alignment. BMC bioinformatics 2010 Sep;11:467–467. https://doi.org/10.1186/1471-2105-11-467.

[207] Smith TF, Waterman MS. Identification of common molecular subsequences. Journal of Molecular Biology 1981;147(1):195 – 197. http://www.sciencedirect.com/science/article/pii/0022283681900875.

[208] Aldinucci M, Meneghin M, Torquati M. Efficient Smith-Waterman on Multi-core with FastFlow. In: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing; 2010. p. 195–199.

[209] Wang L, Chan Y, Duan X, Lan H, Meng X, Liu W. XSW: Accelerating Biological Database Search on Xeon Phi. In: 2014 IEEE International Parallel Distributed Processing Symposium Workshops; 2014. p. 950–957.

[210] Liu Y, Schmidt B. SWAPHI: Smith-waterman protein database search on Xeon Phi coprocessors. In: 2014 IEEE 25th International Con-

ference on Application-Specific Systems, Architectures and Processors 2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors; 2014. p. 184–185. https://doi.org/10.1109/ASAP.2014.6868657.

[211] Rucci E, Garcia Sanchez C, Botella Juan G, Giusti AD, Naiouf M, Prieto-Matias M. SWIMM 2.0: Enhanced Smith–Waterman on Intel's Multicore and Manycore Architectures Based on AVX-512 Vector Extensions. International Journal of Parallel Programming 2019 Apr;47(2):296–316. https://doi.org/10.1007/s10766-018-0585-7.

[212] Baker M, Welch A, Gorentla Venkata M. Parallelizing the Smith-Waterman Algorithm Using OpenSHMEM and MPI-3 One-Sided Interfaces. In: Gorentla Venkata M, Shamis P, Imam N, Lopez MG, editors. OpenSHMEM and Related Technologies. Experiences, Implementations, and Technologies Cham: Springer International Publishing; 2015. p. 178–191.

[213] Khaled H, Faheem HM, Fayez M, Katib I, Aljohani NR. Performance improvement of the parallel smith waterman algorithm implementation using Hybrid MPI-OpenMP model. In: 2016 SAI Computing Conference (SAI) 2016 SAI Computing Conference (SAI); 2016. p. 1232–1234. https://doi.org/10.1109/SAI.2016.7556136.

[214] Liu Y, Huang W, Johnson J, Vaidya S. GPU Accelerated Smith-Waterman. In: Alexandrov VN, van Albada GD, Sloot PMA, Dongarra J, editors. Computational Science – ICCS 2006 Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. p. 188–195.

[215] Liu Y, Schmidt B, Maskell DL. CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Research Notes 2010 Apr;3(1):93. https://doi.org/10.1186/1756-0500-3-93.

[216] de O Sandes EF, Miranda G, de Melo ACMA, Martorell X, Ayguadé E. CUDAlign 3.0: Parallel Biological Sequence Comparison in Large GPU Clusters. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; 2014. p. 160–169. https://doi.org/10.1109/CCGrid.2014.18.

[217] Chenna R, Sugawara H, Koike T, Lopez R, Gibson TJ, Higgins DG, et al. Multiple sequence alignment with the Clustal series of programs.

Nucleic acids research 2003 Jul;31(13):3497–3500. https://doi.org/10.1093/nar/gkg500.

[218] Sandes EFDO, Boukerche A, Melo ACMAD. Parallel Optimal Pairwise Biological Sequence Comparison: Algorithms, Platforms, and Classification. ACM Comput Surv 2016 Mar;48(4). https://doi.org/10.1145/2893488.

[219] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. Journal of Molecular Biology 1990;215(3):403 – 410. http://www.sciencedirect.com/science/article/pii/S0022283605803602.

[220] Ye W, Chen Y, Zhang Y, Xu Y. H-BLAST: a fast protein sequence alignment toolkit on heterogeneous computers with GPUs. Bioinformatics 2017 01;33(8):1130–1138. https://doi.org/10.1093/bioinformatics/btw769.

[221] Li YC, Lu YC. BLASTP-ACC: Parallel Architecture and Hardware Accelerator Design for BLAST-Based Protein Sequence Alignment. IEEE Transactions on Biomedical Circuits and Systems 2019;13(6):1771–1782. https://doi.org/10.1109/TBCAS.2019.2943539.

[222] Mathog DR. Parallel BLAST on split databases. Bioinformatics 2003 09;19(14):1865–1866. https://doi.org/10.1093/bioinformatics/btg250.

[223] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics (Oxford, England) 2009 Aug;25(16):2078–2079. https://doi.org/10.1093/bioinformatics/btp352.

[224] Follia L, Tordini F, Pernice S, Romano G, Piaggeschi GB, Ferrero G. ParallNormal: An Efficient Variant Calling Pipeline for Unmatched Sequencing Data. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP) 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP); 2018. p. 423–429. https://doi.org/10.1109/PDP2018.2018.00074.

[225] Hsu Y, Matsuoka M, Jung N, Matsumoto Y, Motooka D, Nakamura S. [Regular Paper] A High-Performance Sequence Analysis Engine for Shotgun Metagenomics through GPU Acceleration. In: 2018 IEEE

18th International Conference on Bioinformatics and Bioengineering (BIBE) 2018 IEEE 18th International Conference on Bioinformatics and Bioengineering (BIBE); 2018. p. 54–60. https://doi.org/10.1109/BIBE.2018.00018.

[226] Matias Rodrigues JF, von Mering C. HPC-CLUST: distributed hierarchical clustering for large sets of nucleotide sequences. Bioinformatics (Oxford, England) 2014 Jan;30(2):287–288. https://doi.org/10.1093/bioinformatics/btt657.

[227] Chiara M, Gioiosa S, Chillemi G, D'Antonio M, Flati T, Picardi E, et al. CoVaCS: a consensus variant calling system. BMC genomics 2018 Feb;19(1):120–120. https://doi.org/10.1186/s12864-018-4508-1.

[228] Kashyap H, Ahmed HA, Hoque N, Roy S, Bhattacharyya DK. Big data analytics in bioinformatics: architectures, techniques, tools and issues. Network Modeling Analysis in Health Informatics and Bioinformatics 2016 Sep;5(1):28. https://doi.org/10.1007/s13721-016-0135-4.

[229] Jansson K, Sundell H, Boström H. gpuRF and gpuERT: Efficient and Scalable GPU Algorithms for Decision Tree Ensembles. In: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops 2014 IEEE International Parallel & Distributed Processing Symposium Workshops; 2014. p. 1612–1621. https://doi.org/10.1109/IPDPSW.2014.180.

[230] You Y, Fu H, Song SL, Randles A, Kerbyson D, Marquez A, et al. Scaling Support Vector Machines on modern HPC platforms. Journal of Parallel and Distributed Computing 2015;76:16 – 31. http://www.sciencedirect.com/science/article/pii/S0743731514001683, special Issue on Architecture and Algorithms for Irregular Applications.

[231] Li Q, Salman R, Test E, Strack R, Kecman V. GPUSVM: a comprehensive CUDA based support vector machine package. Open Computer Science 01 Dec 2011;1(4):387 – 405. https://www.degruyter.com/view/journals/comp/1/4/article-p387.xml.

[232] Kamburugamuve S, Wickramasinghe P, Ekanayake S, Fox GC. Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink. The International Journal of High Performance Computing Applications 2018;32(1):61–73. https://doi.org/10.1177/1094342017712976.

[233] Schemala Dd, Schlesinger D, Winkler P, Herold H, Meinel G, Semantic segmentation of settlement patterns in gray-scale map images using RF and CRF within an HPC environment; 2016. `http://proceedings.utwente.nl/420/`.

[234] Mei K, Dong P, Lei H, Fan J. A distributed approach for large-scale classifier training and image classification. Neurocomputing 2014;144:304 – 317. `http://www.sciencedirect.com/science/article/pii/S0925231214006456`.

[235] Petrini A, Mesiti M, Schubach M, Frasca M, Danis D, Re M, et al. parSMURF, a high-performance computing tool for the genome-wide detection of pathogenic variants. GigaScience 2020 05;9(5). `https://doi.org/10.1093/gigascience/giaa052`, giaa052.

[236] Ashley EA. Towards precision medicine. Nature Reviews Genetics 2016;17:507–522.

[237] Fogel A, Kvedar JC. Artificial intelligence powers digital medicine. Nature Digital Medicine 2018;1(1).

[238] Leung MKK, Delong A, Alipanahi B, Frey BJ. Machine Learning in Genomic Medicine: A Review of Computational Problems and Data Sets. Proceedings of the IEEE 2016;104:176–197.

[239] Ward LD, Kellis M. Interpreting noncoding genetic variation in complex traits and human disease. Nature biotechnology 2012;30(11):1095.

[240] Veltman J, Lupski J. From genes to genomes in the clinic. Genome Medicine 2015 Jul;7.

[241] Abecasis GR, Auton A, Brooks LD, DePristo MA, Durbin RM, Handsaker RE, et al. An integrated map of genetic variation from 1,092 human genomes. Nature 2012 Nov;491(7422):56–65. `http://dx.doi.org/10.1038/nature11632`.

[242] Turnbull C, Scott RH, Thomas E, Jones L, Murugaesu N, Pretty FB, et al. The 100 000 Genomes Project: bringing whole genome sequencing to the NHS. BMJ 2018;361. `https://www.bmj.com/content/361/bmj.k1687`.

[243] Kumar P, Henikoff S, Ng P. Predicting the effects of coding non-synonymous variants on protein function using the SIFT algorithm. Nat Protoc 2009;4(7):1073–81.

[244] Adzhubei I, Jordan DM, Sunyaev SR. Predicting functional effect of human missense mutations using PolyPhen-2. Current protocols in human genetics 2013;76(1):7–20.

[245] Bendl J, Musil M, Stourac J, Zendulka J, Damborsky J, Brezovsky J. PredictSNP2: A unified platform for accurately evaluating SNP effects by exploiting the different characteristics of variants in distinct genomic regions. PLOS Computational Biology 2016;e100496.

[246] Edwards SL, Beesley J, French JD, Dunning AM. Beyond GWASs: illuminating the dark road from association to function. American Journal of Human Genetics 2013;93 5:779–97.

[247] Zhou J, Theesfeld CL, Yao K, Chen KM, Wong AK, Troyanskaya OG. Deep learning sequence-based ab initio prediction of variant effects on expression and disease risk. Nature Genetics 2018;50:1171–1179.

[248] Rojano E, Seoane P, Ranea JAG, Perkins JR. Regulatory variants: from detection to predicting impact. Briefings in Bioinformatics 2018 06;20(5):1639–1654. https://doi.org/10.1093/bib/bby039.

[249] Telenti A, Lippert C, Chang PC, DePristo M. Deep learning of genomic variation and regulatory network data. Human molecular genetics 2018;27(R1):R63–R71.

[250] He H, Garcia EA. Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering 2009 Sep;21(9):1263–1284. https://doi.org/10.1109/TKDE.2008.239.

[251] Breiman L. Random forests. Machine Learning 2001;45(1):5–32.

[252] Dudley J, Shameer K, Kalari K, Tripathi L, Sowdhamini R. Interpreting functional effects of coding variants: challenges in proteome-scale prediction, annotation and assessment. Briefings in Bioinformatics 2015 10;17(5):841–862.

[253] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic Minority Over-sampling Technique. J Artif Int Res 2002 Jun;16(1):321–357.

[254] Petrini A, Schubach M, Re M, Frasca M, Mesiti M, Grossi G, et al. Parameters tuning boosts hyperSMURF predictions of rare deleterious non-coding genetic variants. PeerJ Preprints 2017;5(e3185v1).

[255] Snoek J, Larochelle H, Adams RP. Practical Bayesian Optimization of Machine Learning Algorithms. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2 NIPS'12, USA: Curran Associates Inc.; 2012. p. 2951–2959.

[256] Snoek J, Larochelle H, Adams RP, GitHub, editor, Spearmint-lite Bayesian optimizer code repository. GitHub; 2012. `https://github.com/JasperSnoek/spearmint`, accessed 04/07/2019.

[257] Consortium TC, Consortium TC, editor, CINECA High Performance Computing homepage. The CINECA Consortium; 2018. `https://www.cineca.it/en/hpc`, accessed 04/07/2019.

[258] Petrini A, GitHub, editor, parSMURF GitHub code repository. GitHub; 2018. `https://github.com/AnacletoLAB/parSMURF`, accessed 04/07/2019.

[259] Paten B, Herrero J, Beal K, Fitzgerald S, Birney E. Enredo and Pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. Genome Res 2008 Nov;18(11):1814–1828. `http://dx.doi.org/10.1101/gr.076554.108`.

[260] Hindorff LA, Sethupathy P, Junkins HA, Ramos EM, Mehta JP, Collins FS, et al. Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. Proceedings of the National Academy of Sciences 2009;106(23):9362–9367. `http://www.pnas.org/content/106/23/9362`.

[261] Grama A, Karypis G, Kumar V, Gupta A. Introduction to Parallel Computing (2nd Edition). 2 ed. Addison Wesley; 2003.

[262] Aljabri M, Trinder PW. Performance comparison of OpenMP and MPI for a concordance benchmark. In: Proceedings of the Saudi Scientific International Conference 2012, London, UK, 11-14 Oct 2012 London, UK: Saudi Scientific International Conference; 2012.p. 22. `http://eprints.gla.ac.uk/78934/`, accessed 07/15/2018.

[263] Mallón DA, Taboada GL, Teijeiro C, Touriño J, Fraguela BB, Gómez A, et al. Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures. In: Ropo M, Westerholm J, Dongarra J, editors. Recent Advances in Parallel Virtual Machine and Message Passing Interface Springer Berlin Heidelberg; 2009. p. 174–184.

[264] Dorta I, León C, Rodríguez C. A comparison between MPI and OpenMP Branch-and-Bound Skeletons. In: High-Level Parallel Programming Models andSupportive Environments, 2003. Proceedings. Eighth International Workshop on IEEE; 2003. p. 66–73.

[265] Krawezik G. Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors. In: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures ACM; 2003. p. 118–127.

[266] Luecke G, Weiss O, Kraeva M, Coyle J, Hoekstra J. Performance Analysis of Pure MPI versus MPI+ OpenMP for Jacobi Iteration and a 3D FFT on the Cray XT5. In: University IS, editor. Iowa State University Graduate Thesis and Dissertations 2012 Citeseer, Iowa State University Digital Repository; 2010. p. 30.

[267] Davis J, Goadrich M. The Relationship Between Precision-Recall and ROC Curves. In: Proceedings of the 23rd International Conference on Machine Learning ICML '06, New York, NY, USA: ACM; 2006. p. 233–240.

[268] for Biotechnology Information NC, for Biotechnology Information TNC, editor, Archive of common human variants in VCF format. The National Institute of Health; 2018. `ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606/VCF/common_all_20180418.vcf.gz`, accessed 10/02/2019.

[269] Jäger M, Wang K, Bauer S, Smedley D, Krawitz P, Robinson PN. Jannovar: a java library for exome annotation. Human mutation 2014;35 5:548–55.

[270] Latchman DS. Transcription factors: An overview. The International Journal of Biochemistry and Cell Biology 1997;29(12):1305 – 1312. `http://www.sciencedirect.com/science/article/pii/S135727259700085X`.

[271] Mora A, Sandve GK, Gabrielsen OS, Eskeland R. In the loop: promoter-enhancer interactions and bioinformatics. Briefings in bioinformatics 2016 11;17(6):980–995.

[272] Lambert SA, Jolma A, Campitelli LF, Das PK, Yin Y, Albu M, et al. The human transcription factors. Cell 2018;172(4):650–665.

[273] Javierre BM, Burren OS, Wilder SP, Kreuzhuber R, Hill SM, Sewitz S, et al. Lineage-Specific Genome Architecture Links Enhancers and Non-coding Disease Variants to Target Gene Promoters. Cell 2016 Nov;167(5):1369–1384.e19. https://doi.org/10.1016/j.cell.2016.09.037.

[274] Bernstein BE, Stamatoyannopoulos JA, Costello JF, Ren B, Milosavljevic A, Meissner A, et al. The NIH Roadmap Epigenomics Mapping Consortium. Nature Biotechnology 2010 Oct;28(10):1045–1048. https://doi.org/10.1038/nbt1010-1045.

[275] Shen Y, Yue F, McCleary DF, Ye Z, Edsall L, Kuan S, et al. A map of the cis-regulatory sequences in the mouse genome. Nature 2012 Aug;488(7409):116–120. https://doi.org/10.1038/nature11243.

[276] Zhu J, Adli M, Zou JY, Verstappen G, Coyne M, Zhang X, et al. Genome-wide Chromatin State Transitions Associated with Developmental and Environmental Cues. Cell 2013;152(3):642 – 654. http://www.sciencedirect.com/science/article/pii/S0092867412015553.

[277] Lizio M, Harshbarger J, Shimoji H, Severin J, Kasukawa T, Sahin S, et al. Gateways to the FANTOM5 promoter level mammalian expression atlas. Genome biology 2015 Jan;16(1):22–22. https://pubmed.ncbi.nlm.nih.gov/25723102.

[278] Consortium RE. Integrative analysis of 111 reference human epigenomes. Nature 2015 Feb;518(7539):317–330. https://doi.org/10.1038/nature14248.

[279] Kwasnieski JC, Fiore C, Chaudhari HG, Cohen BA. High-throughput functional testing of ENCODE segmentation predictions. Genome research 2014 Oct;24(10):1595–1602. https://pubmed.ncbi.nlm.nih.gov/25035418.

[280] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature 2015 May;521(7553):436–444. https://doi.org/10.1038/nature14539.

[281] Bengio Y, Courville A, Vincent P. Representation Learning: A Review and New Perspectives. IEEE Trans Pattern Anal Mach Intell 2013 Aug;35(8):1798–1828. https://doi.org/10.1109/TPAMI.2013.50.

[282] Hinton GE, Salakhutdinov RR. Reducing the Dimensionality of Data with Neural Networks. Science 2006;313(5786):504–507. https://science.sciencemag.org/content/313/5786/504.

[283] Park Y, Kellis M. Deep learning for regulatory genomics. Nature Biotechnology 2015 Aug;33(8):825–826. https://doi.org/10.1038/nbt.3313.

[284] Andersson R, Gebhard C, Miguel-Escalada I, Hoof I, Bornholdt J, Boyd M, et al. An atlas of active enhancers across human cell types and tissues. Nature 2014 Mar;507(7493):455–461. https://doi.org/10.1038/nature12787.

[285] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks 2015 Jan;61:85–117. http://www.sciencedirect.com/science/article/pii/S0893608014002135.

[286] Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks 1988 Jan;1(2):119–130. http://www.sciencedirect.com/science/article/pii/0893608088900147.

[287] Maaten Lvd, Hinton G. Visualizing data using t-SNE. Journal of machine learning research 2008;9(Nov):2579–2605.

[288] Hierlemann A, Schweizer-Berberich M, Weimar U, Kraus G, Pfau A, Göpel W. Pattern recognition and multicomponent analysis. Sensors update 1996;2(1):119–180.

[289] Bergstra J, Bengio Y. Random Search for Hyper-parameter Optimization. Journal of Machine Learning Research 2012 Feb;13(1):281–305. http://dl.acm.org/citation.cfm?id=2503308.2188395.

[290] Hazan E, Klivans A, Yuan Y. Hyperparameter optimization. 6th International Conference on Learning Representations, ICLR 2018; 2018. http://www.scopus.com/inward/record.url?scp=85083950858&partnerID=8YFLogxK.

[291] Swersky K, Snoek J, Adams RP. Multi-Task Bayesian Optimization. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ, editors. Advances in Neural Information Processing Systems 26 Curran Associates, Inc.; 2013.p. 2004–2012. http://papers.nips.cc/paper/5086-multi-task-bayesian-optimization.pdf.

[292] Shahriari B, Swersky K, Wang Z, Adams RP, de Freitas N. Taking the Human Out of the Loop: A Review of Bayesian Optimization. Proceedings of the IEEE 2016 jan;104(1):148–175.

[293] Nair V, Hinton GE. Rectified Linear Units Improve Restricted Boltzmann Machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning ICML'10, Madison, WI, USA: Omnipress; 2010. p. 807–814.

[294] Bewick V, Cheek L, Ball J. Statistics review 13: receiver operating characteristic curves. Critical care (London, England) 2004 Dec;8(6):508–512. https://pubmed.ncbi.nlm.nih.gov/15566624.

[295] Boyd K, Eng KH, Page CD. Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals. In: Blockeel H, Kersting K, Nijssen S, Železný F, editors. Machine Learning and Knowledge Discovery in Databases Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 451–466.

[296] Fawcett T. An introduction to ROC analysis. Pattern Recognition Letters 2006 Jun;27(8):861–874. http://www.sciencedirect.com/science/article/pii/S016786550500303X.

[297] Wilcoxon F. Individual Comparisons by Ranking Methods. Biometrics Bulletin 1945 Dec;1(6):80–83.

[298] Pratt JW. Remarks on Zeros and Ties in the Wilcoxon Signed Rank Procedures. Journal of the American Statistical Association 1959 Sep;54(287):655–667.

[299] Derrick B, White P. Comparing two samples from an individual Likert question. International Journal of Mathematics and Statistics 2017;18.

[300] Frasca M, Valentini G. COSNet: An R package for label prediction in unbalanced biological networks. Neurocomputing 2017;237:397 – 400.

[301] Frasca M, Grossi G, Gliozzo J, Mesiti M, Notaro M, Perlasca P, et al. A GPU-based algorithm for fast node label learning in large and unbalanced biomolecular networks. BMC Bioinformatics 2018 Oct;19(10):353. https://doi.org/10.1186/s12859-018-2301-4.

[302] Conte D, Grossi G, Lanzarotti R, Lin J, **A Petrini**. A parallel MCMC algorithm for the Balanced Graph Coloring problem. In: IAPR International workshop on Graph-Based Representation in Pattern Recognition, Tours, France; 2019. p. 8.

[303] Conte D, Grossi G, Lanzarotti R, Lin J, **A Petrini**. Analysis of a parallel MCMC algorithm for graph coloring with nearly uniform balancing. Pattern Recognition Letter 2020;XX(XX):XXX. Submitted for review.

[304] Consortium TU. UniProt: a hub for protein information. Nucleic Acids Research 2014 10;43(D1):D204–D212. `https://doi.org/10.1093/nar/gku989`.

[305] Boutet E, Lieberherr D, Tognolli M, Schneider M, Bansal P, Bridge AJ, et al. Using GenBank. In: Edwards D, editor. UniProtKB/Swiss-Prot, the Manually Annotated Section of the UniProt KnowledgeBase: How to Use the Entry View New York, NY: Springer New York; 2016. p. 23–54.

[306] Friedberg I. Automated protein function prediction-the genomic challenge. Brief Bioinformatics 2006;7:225–242.

[307] Szklarczyk D, et al. STRING v10: protein–protein interaction networks, integrated over the tree of life. Nucleic Acids Research 2015;43(D1):D447–D452.

[308] Oliver S. Guilt-by-association goes global. Nature 2000 Feb;403(6770):601–602. `https://doi.org/10.1038/35001165`.

[309] Chaudhari G, Avadhanula V, Sarawagi S. A Few Good Predictions: Selective Node Labeling in a Social Network. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining WSDM '14, New York, NY, USA: ACM; 2014. p. 353–362.

[310] Zhu X, Ghahramani Z, Lafferty J. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In: Proceedings of the Twentieth International Conference on International Conference on Machine Learning ICML'03, AAAI Press; 2003. p. 912–919.

[311] Tsuda K, Shin H, Scholkopf B. Fast protein classification with multiple networks. Bioinformatics 2005 09;21(suppl-2):ii59–ii65. `https://doi.org/10.1093/bioinformatics/bti1110`.

[312] Zhou D, Bousquet O, Lal T, Weston J, Olkopf B. Learning with Local and Global Consistency. Advances in Neural Information Processing Systems 16 2004 03;16.

[313] Mostafavi S, Ray D, Warde-Farley D, Grouios C, Morris Q. GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. Genome biology 2008;9 Suppl 1(Suppl 1):S4–S4. https://pubmed.ncbi.nlm.nih.gov/18613948.

[314] Vazquez A, Flammini A, Maritan A, Vespignani A. Global protein function prediction from protein-protein interaction networks. Nature Biotechnology 2003 Jun;21(6):697–700. https://doi.org/10.1038/nbt825.

[315] Karaoz U, Murali TM, Letovsky S, Zheng Y, Ding C, Cantor CR, et al. Whole-genome annotation by using evidence integration in functional-linkage networks. Proceedings of the National Academy of Sciences of the United States of America 2004 Mar;101(9):2888–2893. https://pubmed.ncbi.nlm.nih.gov/14981259.

[316] Bertoni A, Frasca M, Valentini G. COSNet: A Cost Sensitive Neural Network for Semi-supervised Learning in Graphs. In: Machine Learning and Knowledge Discovery in Databases – European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I, vol. 6911 of LNAI Springer-Verlag; 2011. p. 219–234.

[317] Frasca M, Bertoni A, Re M, Valentini G. A neural network algorithm for semi-supervised node label learning from unbalanced data. Neural Networks 2013;43:84 – 98. http://www.sciencedirect.com/science/article/pii/S0893608013000361.

[318] Frasca M. Automated gene function prediction through gene multifunctionality in biological networks. Neurocomputing 2015 Aug;162:48–56. http://www.sciencedirect.com/science/article/pii/S0925231215004142.

[319] Frasca M, Bassis S, Valentini G. Learning node labels with multi-category Hopfield networks. Neural Computing and Applications 2016 Aug;27(6):1677–1692. https://doi.org/10.1007/s00521-015-1965-1.

[320] Frasca M, Bertoni A, Sion A. Neural Nets and Surroundings: 22nd Italian Workshop on Neural Nets, WIRN 2012, May 17-19, Vietri sul Mare, Salerno, Italy. In: Apolloni B, Bassis S, Esposito A, Morabito CF, editors. A Neural Procedure for Gene Function Prediction Smart Innovation, Systems and Technologies, Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 179–188.

[321] Chua HN, Sung WK, Wong L. Exploiting indirect neighbours and topological weight to predict protein function from protein–protein interactions. Bioinformatics 2006 04;22(13):1623–1630. https://doi.org/10.1093/bioinformatics/btl145.

[322] Bogdanov P, Singh AK. Molecular Function Prediction Using Neighborhood Features. IEEE/ACM Transactions on Computational Biology and Bioinformatics 2010;7(2):208–217.

[323] Szummer M, Jaakkola T. Partially labeled classification with Markov random walks. In: Advances in Neural Information Processing Systems (NIPS), vol. 14 MIT Press; 2001. p. 945–952.

[324] Azran A. The Rendezvous Algorithm: Multiclass Semi-Supervised Learning with Markov Random Walks. In: Proceedings of the 24th International Conference on Machine Learning ICML '07, New York, NY, USA: Association for Computing Machinery; 2007. p. 49–56. https://doi.org/10.1145/1273496.1273503.

[325] Köhler S, Bauer S, Horn D, Robinson PN. Walking the interactome for prioritization of candidate disease genes. American journal of human genetics 2008 Apr;82(4):949–958. https://pubmed.ncbi.nlm.nih.gov/18371930.

[326] Valentini G, Armano G, Frasca M, Lin J, Mesiti M, Re M. RANKS: a flexible tool for node label ranking and classification in biological networks. Bioinformatics 2016 06;32(18):2872–2874. https://doi.org/10.1093/bioinformatics/btw235.

[327] Frasca M, Bianchi NC. Multitask Protein Function Prediction through Task Dissimilarity. IEEE/ACM Transactions on Computational Biology and Bioinformatics 2019;16(5):1550–1560.

[328] Mislove A, Viswanath B, Gummadi KP, Druschel P. You Are Who You Know: Inferring User Profiles in Online Social Networks. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining WSDM '10, New York, NY, USA: ACM; 2010. p. 251–260. http://doi.acm.org/10.1145/1718487.1718519.

[329] Bhagat S, Cormode G, Muthukrishnan S. Node Classification in Social Networks. Social Network Data Analytics 2011;p. 115–148. http://dx.doi.org/10.1007/978-1-4419-8462-3_5.

[330] Japkowicz N, Stephen S. The Class Imbalance Problem: A Systematic Study. Intell Data Anal 2002 Oct;6(5):429–449.

[331] Mesiti M, Re M, Valentini G. Think globally and solve locally: secondary memory-based network learning for automated multi-species function prediction. GigaScience 2014 Apr;3:5–5. `https://pubmed.ncbi.nlm.nih.gov/24843788`.

[332] NVidia, NVidia, editor, CUDA programming guide. NVidia; 2017. `http://docs.nvidia.com/cuda`.

[333] Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, et al. Gene Ontology: tool for the unification of biology. Nature Genetics 2000 May;25(1):25–29. `https://doi.org/10.1038/75556`.

[334] Petrini A, com G, editor, AnacletoLab official Par-COSNet source code repository hosted on GitHub. GitHub; 2017. `https://github.com/AnacletoLAB/ParCOSNet`.

[335] Chong Y, Ding Y, Yan Q, Pan S. Graph-based semi-supervised learning: A review. Neurocomputing 2020;408:216 – 230. `http://www.sciencedirect.com/science/article/pii/S0925231220304938`.

[336] Bengio Y, Delalleau O, Le Roux N. Label Propagation and Quadratic Criterion. In: Chapelle O, Scholkopf B, Zien A, editors. Semi-Supervised Learning MIT Press; 2006.p. 193–216.

[337] Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences 1982;79(8):2554–2558. `https://www.pnas.org/content/79/8/2554`.

[338] Harish P, Narayanan PJ. Accelerating Large Graph Algorithms on the GPU Using CUDA. In: Aluru S, Parashar M, Badrinath R, Prasanna VK, editors. High Performance Computing – HiPC 2007 Berlin, Heidelberg: Springer Berlin Heidelberg; 2007. p. 197–208.

[339] Luby M. A Simple Parallel Algorithm for the Maximal Independent Set Problem. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing STOC '85, New York, NY, USA: ACM; 1985. p. 1–10.

[340] Frasca M, Bertoni A, Valentini G. UNIPred: Unbalance-Aware Network Integration and Prediction of Protein Functions. Journal of

Computational Biology 2015;22(12):1057–1074. https://doi.org/10.1089/cmb.2014.0110, pMID: 26402488.

[341] Jiang Y, Oron TR, et al. An expanded evaluation of protein function prediction methods shows an improvement in accuracy. Genome Biology 2016;17(1):184. http://dx.doi.org/10.1186/s13059-016-1037-6.

[342] Frasca M, Pavesi G. A neural network based algorithm for gene expression prediction from chromatin structure. In: The 2013 International Joint Conference on Neural Networks (IJCNN) The 2013 International Joint Conference on Neural Networks (IJCNN); 2013. p. 1–8.

[343] Tantipathananandh C, Berger-Wolf T, Kempe D. A Framework for Community Identification in Dynamic Social Networks. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '07, New York, NY, USA: Association for Computing Machinery; 2007. p. 717–726. https://doi.org/10.1145/1281192.1281269.

[344] Mosa MA, Hamouda A, Marei M. Graph coloring and ACO based summarization for social networks. Expert Systems with Applications 2017;74:115 – 126. http://www.sciencedirect.com/science/article/pii/S0957417417300118.

[345] Murad O, Sleit A, Sharaiah A. Improving Friends Matching in Social Networks Using Graph Coloring. Internation journal of computer and technology 2016 Jun;15(8):7028–7034. https://rajpub.com/index.php/ijct/article/view/1503.

[346] Fakhraei S, Foulds J, Shashanka M, Getoor L. Collective Spammer Detection in Evolving Multi-Relational Social Networks. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '15, New York, NY, USA: Association for Computing Machinery; 2015. p. 1769–1778. https://doi.org/10.1145/2783258.2788606.

[347] Lupaşcu CA, Tegolo D, Bellavia F, Valenti C. Semi-automatic registration of retinal images based on line matching approach. In: Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems, IEEE; 2013. p. 453–456. https://doi.org/10.1109%2Fcbms.2013.6627839.

[348] Cuculo V, Lanzarotti R, Boccignone G. Using Sparse Coding for Landmark Localization in Facial Expressions. In: 2014 5th European Workshop on Visual Information Processing (EUVIP); 2014. p. 1–6.

[349] Tsolkas D, Liotou E, Passas N, Merakos L. A graph-coloring secondary resource allocation for D2D communications in LTE networks. In: 2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD) 2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD); 2012. p. 56–60. https://doi.org/10.1109/CAMAD.2012.6335378.

[350] Comi P, Crosta PS, Beccari M, Paglierani P, Grossi G, Pedersini F, et al. Hardware-accelerated high-resolution video coding in Virtual Network Functions. In: Eu. Conf. on Network and Communication; 2016. p. 32–36.

[351] Janis P, Chia-Hao Y, Doppler K, Ribeiro C, Wijting C, Klaus H, et al. Device-to-device communication underlaying cellular communications systems. Int Jour of Comm, Network and System Sciences 2009;2-3.

[352] Coleman TF, Moré JJ. Estimation of sparse hessian matrices and graph coloring problems. Mathematical Programming 1984 Oct;28(3):243–270. https://doi.org/10.1007/BF02612334.

[353] Deveci M, Boman EG, Devine KD, Rajamanickam S. Parallel Graph Coloring for Manycore Architectures. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2016. p. 892–901. https://doi.org/10.1109/IPDPS.2016.54.

[354] Chen X, Li P, Fang J, Tang T, Wang Z, Yang C. Efficient and high-quality sparse graph coloring on GPUs. Concurrency and Computation: Practice and Experience 2017 May;29(10):e4064. https://doi.org/10.1002/cpe.4064.

[355] Jones MT, Plassmann PE. A Parallel Graph Coloring Heuristic. SIAM Journal on Scientific Computing 1993 May;14(3):654–669. https://doi.org/10.1137/0914041.

[356] Gjertsen RK, Jones MT, Plassmann PE. Parallel heuristics for improved, balanced graph colorings. J Par and Dist Comput 1996;37:171–186.

[357] Boman EG, Bozdağ D, Catalyurek U, Gebremedhin AH, Manne F. A Scalable Parallel Graph Coloring Algorithm for Distributed Memory Computers. In: Cunha JC, Medeiros PD, editors. Euro-Par 2005 Parallel Processing Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 241–251.

[358] Meyer W. Equitable coloring. Amer Math Monthly 1973;80:920–922.

[359] Bertoni A, Goldwurm M, Lin J, Saccà F. Size constrained distance clustering: separation properties and some complexity results. Fundamenta Informaticae 2012;115(1):125–139.

[360] Lu H, Halappanavar M, Chavarría-Miranda D, Gebremedhin A, Kalyanaraman A. Balanced Coloring for Parallel Computing Applications. In: 2015 IEEE International Parallel and Distributed Processing Symposium 2015 IEEE International Parallel and Distributed Processing Symposium; 2015. p. 7–16. https://doi.org/10.1109/IPDPS.2015.113.

[361] Jerrum M. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. Random Structures & Algorithms 1995;7(2):157–165. https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.3240070205.

[362] Hastings W. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 1970;57(1):97–109.

[363] Voss J. An introduction to statistical computing: a simulation-based approach. John Wiley & Sons; 2013.

[364] Daskalakis C, Kamath G, Tzamos C. On the Structure, Covering, and Learning of Poisson Multinomial Distributions. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science 2015 IEEE 56th Annual Symposium on Foundations of Computer Science; 2015. p. 1203–1217. https://doi.org/10.1109/FOCS.2015.77.

[365] Erdos P, Rényi A. 2. In: On the evolution of random graphs, vol. 9781400841356 Princeton University Press; 2011. p. 38–82.

[366] Rossi RA, Ahmed NK. An Interactive Data Repository with Visual Analytics. SIGKDD Explor 2016;17(2):37–41.

[367] Mood AM, Graybill FA, Boes DC. Introduction to the Theory of Statistics. McGraw-Hill; 1973.

[368] Cochran WG. The χ^2 Test of Goodness of Fit. Ann Math Statist 1952 09;23(3):315–345. https://doi.org/10.1214/aoms/1177729380.

[369] Brodtkorb E, Strand J, Backe PH, Lund AM, Bjørås M, Rootwelt T, et al. Four novel mutations identified in Norwegian patients result in intermittent maple syrup urine disease when combined with the R301C mutation. Molecular Genetics and Metabolism 2010;100(4):324 – 332. http://www.sciencedirect.com/science/article/pii/S1096719210001812.

[370] Al-Shali K, Cao H, Knoers N, Hermus AR, Tack CJ, Hegele RA. A Single-Base Mutation in the Peroxisome Proliferator-Activated Receptor γ4 Promoter Associated with Altered in Vitro Expression and Partial Lipodystrophy. The Journal of Clinical Endocrinology & Metabolism 2004 11;89(11):5655–5660. https://doi.org/10.1210/jc.2004-0280.

[371] Funnell APW, Wilson MD, Ballester B, Mak KS, Burdach J, Magan N, et al. A CpG Mutational Hotspot in a ONECUT Binding Site Accounts for the Prevalent Variant of Hemophilia B Leyden. The American Journal of Human Genetics 2013;92(3):460 – 467. http://www.sciencedirect.com/science/article/pii/S0002929713000748.

[372] Roessler E, Hu P, Hong SK, Srivastava K, Carrington B, Sood R, et al. Unique Alterations of an Ultraconserved Non-Coding Element in the 3'UTR of ZIC2 in Holoprosencephaly. PLOS ONE 2012 07;7(7):1–6. https://doi.org/10.1371/journal.pone.0039026.

[373] Haghighi K, Chen G, Sato Y, Fan GC, He S, Kolokathis F, et al. A human phospholamban promoter polymorphism in dilated cardiomyopathy alters transcriptional regulation by glucocorticoids. Human Mutation 2008;29(5):640–647. https://onlinelibrary.wiley.com/doi/abs/10.1002/humu.20692.

[374] Sabatelli M, Moncada A, Conte A, Lattante S, Marangi G, Luigetti M, et al. Mutations in the 3' untranslated region of FUS causing FUS overexpression are associated with amyotrophic lateral sclerosis. Human Molecular Genetics 2013 07;22(23):4748–4755. https://doi.org/10.1093/hmg/ddt328.

[375] Fang Q, Chen S, Wang Y, Jiang S, Zhang R, Hu C, et al. Functional analyses of the mutation nt-128 T->G in the hepatocyte nuclear factor-1α promoter region in Chinese diabetes pedigrees. Di-

abetic Medicine 2012;29(11):1456–1464. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1464-5491.2012.03626.x.

[376] Petrini A, Mesiti M, Schubach M, Frasca M, Danis D, Re M, et al., Foundation TOS, editor, Datasets used for the assessment of prediction quality and scalability, hosted at the Open Science Foundation project page. The Open Science Foundation; 2019. https://osf.io/m8e6z/?view_only=e746468f88654f0c954b4b4bee2f7c4d, accessed 04/10/2019, DOI 10.17605/OSF.IO/M8E6Z.

[377] Petrini A, Mesiti M, Schubach M, Frasca M, Danis D, Re M, et al., GigaScience, editor, Supporting data for "parSMURF, a High Performance Computing tool for the genome-wide detection of pathogenic variants", GigaScience Database 2020. GigaScience; 2020. http://dx.doi.org/10.5524/100743, dOI:10.5524/100743.

Appendices

Appendix A - parSMURF supplementary Figures and Tables

Here we report all the supplementary materials for chapter 3.

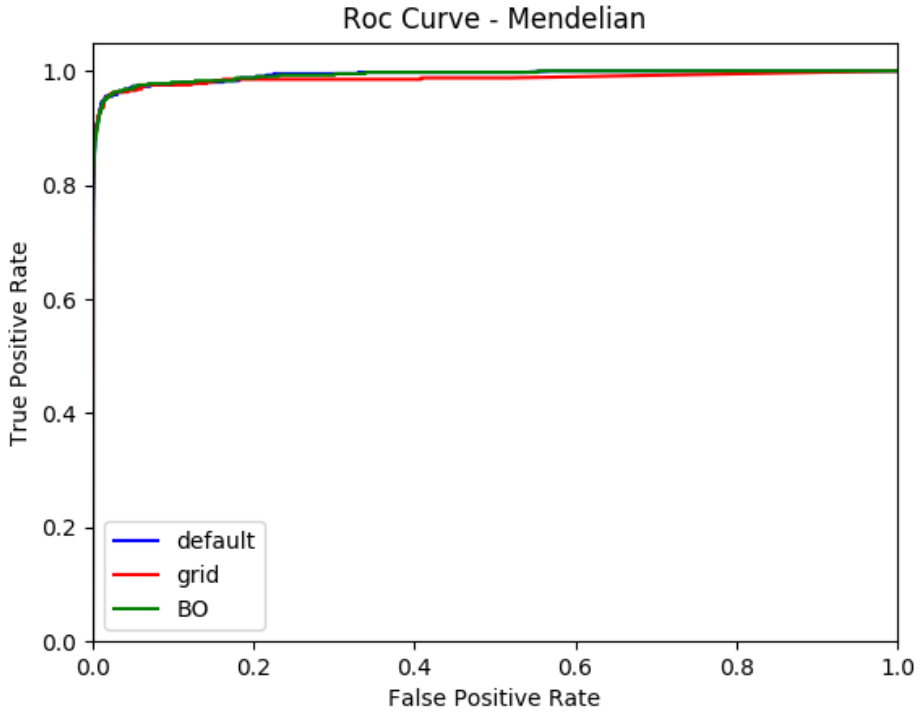


Figure 1: Plot of Receiver Operating Characteristic curve of the predictions for the Mendelian dataset using 3 sets of hyper-parameters: default (blue), best from the grid search (red) and best from the Bayesian optimizer (green)

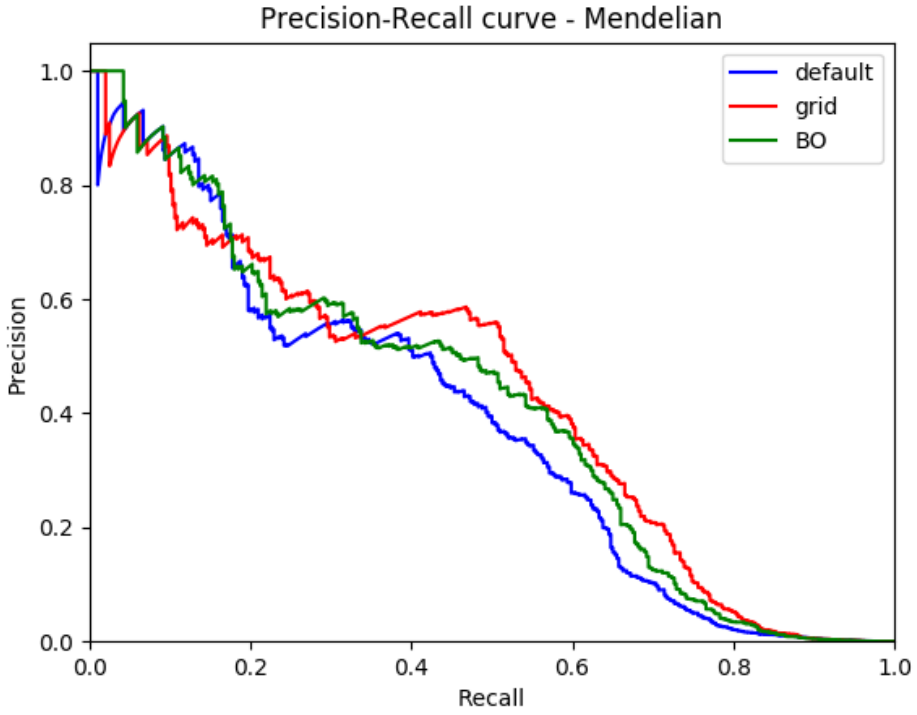


Figure 2: Plot of Precision-Recall curve of the predictions for the Mendelian dataset using 3 sets of hyper-parameters: default (blue), best from the grid search (red) and best from the Bayesian optimizer (green)

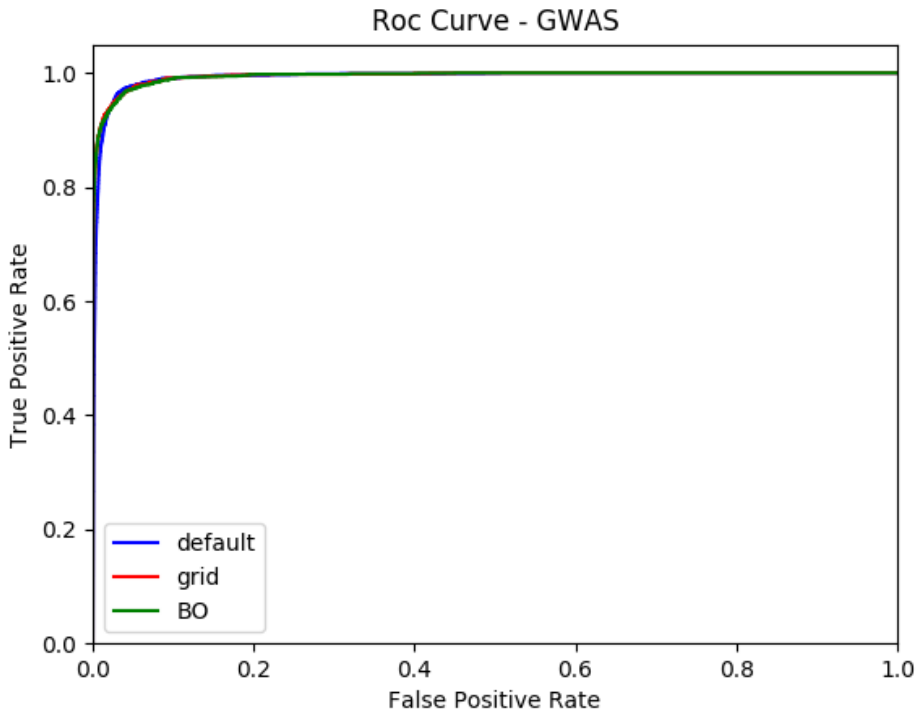


Figure 3: Plot of Receiver Operating Characteristic curve of the predictions for the GWASn dataset using 3 sets of hyper-parameters: default (blue), best from the grid search (red) and best from the Bayesian optimizer (green)

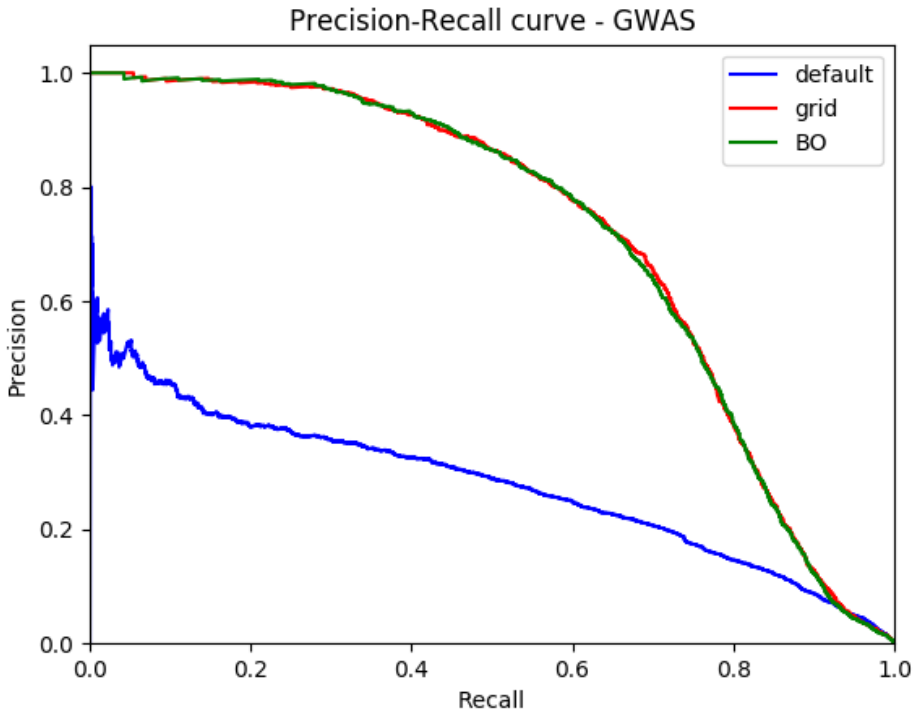


Figure 4: Plot of Precision-Recall curve of the predictions for the GWAS dataset using 3 sets of hyper-parameters: default (blue), best from the grid search (red) and best from the Bayesian optimizer (green)

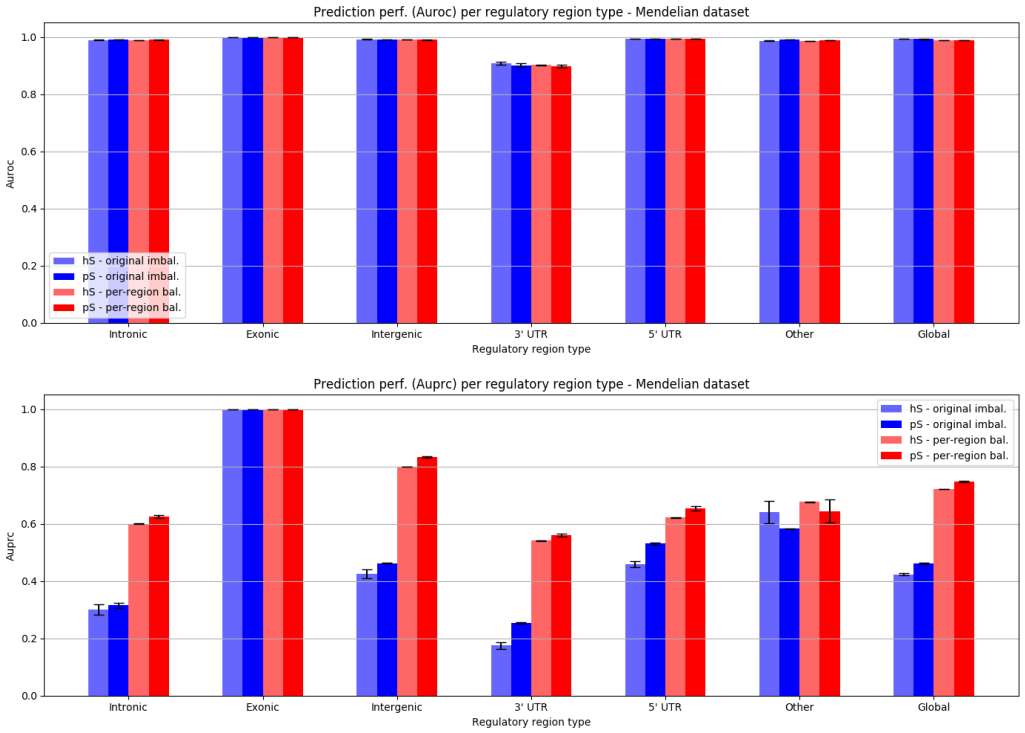


Figure 5: Prediction performances (AUROC and AUPRC) for the Mendelian dataset, with both the Original imbalanced Mendelian data set and with the separated "per-region balanced" Mendelian data used for validation using *hyperSMURF*(hS) and *parSMURF*(pS) with Bayesian optimization hyper-ensembles (set of hyper-parameters found by the Bayesian optimizer: `nParts`: 36 - `fp`: 3 - `ratio`: 5 - `k`: 5 - `numTrees`: 55 - `mtry`: 5). Each group shows prediction performances divided by regulatory region type. "Global" group shows prediction performances of the entire dataset, disregarding the regulatory region type. AUROC and AUPRC have been averaged across "cytoband-aware" 10-fold cross validation. Results have been averaged across 10 repetitions of the same experimental setup (error bars represent standard deviation). "Original imbalanced" and "Per-region balanced" stands respectively for the original imbalanced setting and the balanced setting where the same ratio of positives versus negatives in both training and test sets is maintained in each regulatory region.

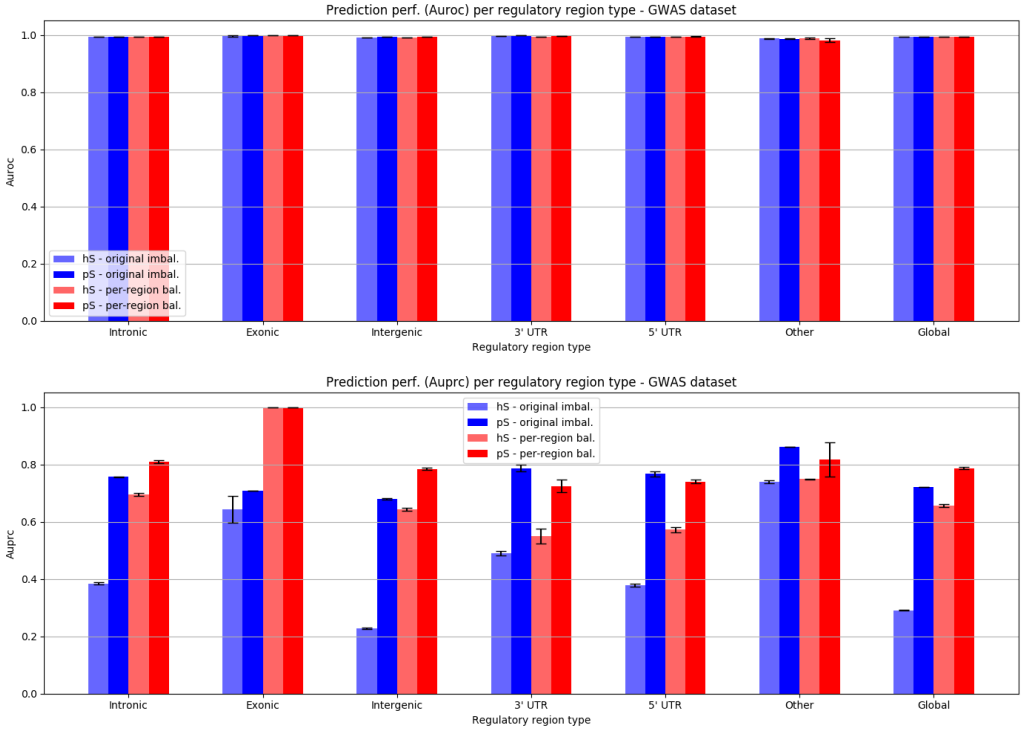


Figure 6: Prediction performances (AUROC and AUPRC) for the GWAS dataset, with both the Original imbalanced GWAS data set and with the separated "per-region balanced" GWAS data used for validation using *hyperSMURF*(hS) and *parSMURF*(pS) with Bayesian optimization hyper-ensembles (set of hyper-parameters found by the Bayesian optimizer: `nParts`: 10 - `fp`: 2 - `ratio`: 5 - `k`: 5 - `numTrees`: 50 - `mtry`: 30). Each group shows prediction performances divided by regulatory region type. "Global" group shows prediction performances of the entire dataset, disregarding the regulatory region type. AUROC and AUPRC have been averaged across "cytoband-aware" 10-fold cross validation. Results have been averaged across 5 repetitions of the same experimental setup (error bars represent standard deviation). "Original imbalanced" and "Per-region balanced" stands respectively for the original imbalanced setting and the balanced setting where the same ratio of positives versus negatives in both training and test sets is maintained in each regulatory region.

Table 1: Optimal sets of hyper-parameters returned by the optimizers embedded in *parSMURF*while training the model with the Mendelian dataset. Refer to Table 1 in the main paper for a brief explanation of each parameter. Default values are those used with *hyperSMURF*. As Grid and Bayesian optimizer returns the set of best hyper-parameters for each fold of the cross-validation, optimal values have been computed by averaging each parameter over the folds.

	nParts	**fp**	**ratio**	**k**	**nTrees**	**mtry**
Default	100	2	3	5	10	5
Grid search	260	7	10	5	50	4
Bayesian optimizer	74	8	10	5	50	5

Table 2: Optimal sets of hyper-parameters returned by the optimizers embedded in *parSMURF*while training the model with the GWAS dataset. Refer to Table 1 in the main paper for a brief explanation of each parameter. Default values are those used with *hyperSMURF*. As Grid and Bayesian optimizer returns the set of best hyper-parameters for each fold of the cross-validation, optimal values have been computed by averaging each parameter over the folds.

	nParts	**fp**	**ratio**	**k**	**nTrees**	**mtry**
Default	100	2	3	5	10	5
Grid search	10	5	10	5	100	30
Bayesian optimizer	10	4	10	5	100	30

Table 3: Spearman correlation between HyperSMURF and parSMURF scores for each of the 26 features of the Mendelian dataset.

Feature name	**Hyper SMURF**	**parSMURF default**	**parSMURF grid**	**parSMURF BO**
CpGobsExp	0.13968	0.13819	0.13893	0.13132
CpGperCpG	0.13969	0.13819	0.13893	0.13132
CpGperGC	0.13968	0.13819	0.13893	0.13132
DGVCount	0.03413	0.04442	0.05647	0.06046
DnaseClusteredHyp	0.32951	0.32044	0.24895	0.28336
DnaseClusteredScore	0.32908	0.32009	0.24869	0.28315
EncH3K27Ac	0.30594	0.30151	0.24630	0.30958
EncH3K4Me1	0.31055	0.30229	0.25376	0.31746
EncH3K4Me3	0.23845	0.23367	0.19111	0.22128
GCContent	0.20083	0.18597	0.15363	0.19790
GerpRS	0.35444	0.35057	0.31584	0.30759
GerpRSpv	0.18839	0.18847	0.15510	0.15381
ISCApath	0.00046	0.00916	−0.01227	0.00711
commonVar	−0.18408	−0.20035	−0.13659	−0.16558
dbVARCount	0.03413	0.04442	0.05647	0.06046
fantom5Perm	0.06291	0.06130	0.05101	0.05391
fantom5Robust	0.06436	0.06269	0.05217	0.05513
fracRareCommon	0.18258	0.20198	0.15003	0.17902
mamPhastCons46way	0.14808	0.14348	0.15451	0.16040
mamPhyloP46way	0.37928	0.38087	0.37977	0.43174
numTFBSConserved	0.13326	0.14405	0.14100	0.13788
priPhastCons46way	0.24858	0.25781	0.26809	0.29352
priPhyloP46way	0.33472	0.33576	0.33014	0.37434
rareVar	0.02916	0.03668	0.05208	0.05687
verPhastCons46way	0.15317	0.15002	0.16128	0.16888
verPhyloP46way	0.38423	0.38685	0.38594	0.43717

Table 4: Imbalance of the number of negative and positive examples across different regulatory region types in the Mendelian dataset. In column 4, the ratio between negative and positive samples; in bold, the lowest ratio used to under-sample the negatives for each regulatory region. The last column reports the number of down-sampled negatives for each regulatory region after applying the best ratio found in column 4.

Category	**Number of positive samples**	**Number of negative samples**	**Ratio (neg/pos)**	**Downsampled negative samples**
Intronic	75	5393802	71917	64625
Exonic non coding	2	35058	17529	1723
Intergenic	126	8325494	66075	108570
3' UTR	30	144348	4811	25850
5' UTR	158	843572	5339	136143
Other	15	12925	**861**	12925
			Total negatives:	349836

Table 7: List of newly annotated pathogenic variants used as independent test set to assess the generalization capabilities of *parSMURF*. The first two columns refer to chromosomal coordinates, REF and ALT to the reference and alternative allele (including both SNVs, micro-insertions and micro-deletions), Region the type of regulatory region, Gene the symbolic name of the gene and PMID the PubMed ID of the related publication.

Chr	**Pos**	**Ref**	**Alt**	**Region**	**Gene**	**PMID**
1	155261709	G	A	exonic non-coding	PKLR	18708292
1	155263324	C	T	exonic non-coding	PKLR	26728349
1	155271259	C	G	promoter	PKLR	18708292
1	155271269	CAGAGA	C	promoter	PKLR	26728349

3	10183453	AGCGCG CACGCA GCTCCG CCCCGC GTCCGA CCCGCG GATCCC GCGGC	A	5' UTR	VHL	30006056
3	10183466	C	CTCCGC CCCGCG	5' UTR	VHL	30006056
3	38675715	C	T	promoter	SCN5A	27625342
4	89444948	C	T	promoter	PIGY	26293662
5	131705516	G	A	5' UTR	SLC22A5	31187905
6	100040906	G	T	enhancer	DHS6S1	26507665
6	100040987	G	C	enhancer	DHS6S1	27551809
6	100041040	C	T	enhancer	DHS6S1	26507665
7	19157199	C	T	5' UTR	TWIST1	30040876
7	19157207	G	T	5' UTR	TWIST1	30040876
7	156584142	A	T	enhancer	LMBR1	27592358
7	156585476	C	G	enhancer	SHH	29543231
8	11565816	G	C	5' UTR	GATA4	25099673
8	19796936	A	G	5' UTR	LPL	27578109
8	102505149	GA	G	5' UTR	GRHL2	29499165
8	102505272	CT	C	5' UTR	GRHL2	29499165
8	102505561	G	T	5' UTR	GRHL2	29499165
9	21974830	GCTCCC CGCCGC CCGCTG CCTGCTC	G	5' UTR	CDKN2A	26581427
9	21974847	G	A	5' UTR	CDKN2A	26581427
9	21974868	A	T	5' UTR	CDKN2A	26581427
9	21974916	CCCT	C	5' UTR	CDKN2A	26581427
9	21975024	GAGT	AAAG	5' UTR	CDKN2A	29216274
10	23481888	AGCGGC GGCTGC GGCGGC GCGGCG CCG	A	exonic non-coding	PTF1A	28663161
10	23508442	A	G	enhancer	PTF1A	28663161

11	31828391	ACTT	CA	5' UTR	PAX6	30291432
11	31828391	AC	A	5' UTR	PAX6	30291432
11	31828396	C	T	5' UTR	PAX6	30291432
11	31828461	TAA	T	5' UTR	PAX6	30291432
11	31828474	CT	C	5' UTR	PAX6	30291432
11	31832375	C	T	5' UTR	PAX6	30291432
11	67250359	CCG	C	promoter	AIP	20506337
13	110802675	G	T	3' UTR	COL4A1	27666438
13	110802678	C	T	3' UTR	COL4A1	28369186
13	110802678	C	A	3' UTR	COL4A1	27666438
13	110802679	C	A	3' UTR	COL4A1	27666438
15	23810779	GTCAG	G	promoter	MKRN3	29763903
15	23810849	C	T	promoter	MKRN3	30462148
15	35080829	A	G	3' UTR	ACTC1	27139165
17	10536907	C	T	splicing	MYH3	31491409
17	10559406	C	T	5' UTR	MYH3	31491409
17	41277375	T	A	5' UTR	BRCA1	30075112
17	61996359	G	T	promoter	GH1	27252485
17	70117348	G	A	5' UTR	SOX9	28546996
18	28683379	C	G	promoter	DSC2	27531918
19	2249105	CA	C	promoter	AMH	31238341
19	49468612	GG	CT	5' UTR	FTL	28636169
X	38202566	G	T	enhancer	OTC	29282796
X	38211793	T	G	5' UTR	OTC	29282796
X	38211808	G	A	5' UTR	OTC	29282796
X	38211811	A	G	5' UTR	OTC	29282796
X	38211834	C	T	5' UTR	OTC	29282796
X	38211835	C	T	5' UTR	OTC	29282796
X	38211844	C	A	5' UTR	OTC	29282796
X	70442966	C	CT		GJB1	28283593
X	70443186	G	T		GJB1	28283593
X	70444424	C	T	3' UTR	GJB1	29236290
X	85302634	G	A	promoter	CHM	28271586
X	85302634	G	T	promoter	CHM	28271586
X	138612871	AC	A	promoter	F9	27865967
X	147031110	T	C	3' UTR	FMR1	26554012

Table 5: Imbalance of the number of negative and positive samples across different regulatory region types in the GWAS dataset. In column 4, the ratio between negative and positive samples; in bold, the lowest ratio used to under-sample the negatives for each regulatory region. The last column reports the number of down-sampled negatives for each regulatory region after applying the best ratio found in column 4.

Category	**Number of positive samples**	**Number of negative samples**	**Ratio (neg/pos)**	**Downsampled negative samples**
Intronic	937	539311	575	115166
Exonic non coding	2	3511	1755	245
Intergenic	891	832571	934	109512
3' UTR	46	14470	314	5653
5' UTR	228	84300	370	28023
Other	11	1352	**122**	1352
			Total negatives:	259951

Table 6: Examples of pathogenic Mendelian single nucleotide variants where *parSMURF* sensibly outperformed *hyperSMURF*. The first two columns refer to the chromosomal coordinates of the variant. Ref and Alt to the reference and alternative allele; OMIM to the OMIM code of the associated Mendelian disease; Gene to the symbolic name of the target gene; PMID to the PubMed ID of the related publication; Region to the type of the regulatory region. The two last columns refer to the difference of ranking between respectively *parSMURF* with grid optimization and *parSMURF* with Bayesian optimization with respect to *hyperSMURF*.

Chr	**Position**	**Ref**	**Alt**	**OMIM**	**Gene**	**PMID**	**Region**	**Rank diff. Grid/Default**	**Rank diff. BO/Default**
chr1	100661453	T	G	MIM 248600	DBT	20570198 [369]	3' UTR	2308597	169786
chr3	12421189	A	G	MIM 604367	PPARG	15531525 [370]	Promoter	663054	421027
chrX	138612889	G	A	MIM 306900	F9	23472758 [371]	Promoter	194290	111499
chr13	100638902	A	G	MIM 609637	ZIC2	22859937 [372]	3' UTR	70175	69069
chr6	118869423	A	G	MIM 609909	PLN	18241046 [373]	Promoter	63078	55789
chr16	31202818	G	A	MIM 608030	FUS	23847048 [374]	3' UTR	50539	103623
chr12	121416444	T	G	MIM 600496	HNF1A	22413961 [375]	Promoter	21848	65773

Appendix B - Availability of source code and materials

.1 parSMURF and parSMURF-NG

.1.1 Availability of source code and requirements

Project name: ParSMURF
Project home page: `https://github.com/AnacletoLAB/parSMURF`
GitHub repository: `https://github.com/AnacletoLAB/parSMURF`
SciCrunch RRID: `SCR_017560`
Operating system(s): Linux
Programming language: C++, Python 2.7
Requirements for *parSMURF*[1]: Multi-core x86-64 processor, 512 MB RAM, C++ compiler supporting OpenMP standard.
Requirements for *parSMURF*^{n}: Multi-core x86-64 processor, 1024 MB RAM, implementation of MPI library (i.e. OpenMPI or IntelMPI) installed on each node of the cluster, a reasonably fast interconnecting infrastructure.
License: GNU General Public License v3

Project name: ParSMURF-NG
Project home page: `https://github.com/AnacletoLAB/parSMURF-NG`
GitHub repository: `https://github.com/AnacletoLAB/parSMURF-NG`
Operating system(s): Linux
Programming language: C++, Python 3.5 (only for the Bayesian Optimizer portion of the code)
Requirements: Multi-core x86-64 processor, 1024 MB RAM, implementation of MPI library (i.e. OpenMPI or IntelMPI) installed locally and on each node of the cluster, a reasonably fast interconnecting infrastructure.
License: GNU General Public License v3

.1.2 Availability of supporting data and materials

Datasets used for the assessment of scalability and prediction quality are available at the following page of the Open Science Foundation project: [376].

Supplementary Information with detailed experimental results are downloadable from the GigaScience website.

Supporting data is available at GigaDB data repository [377].

.2 Par-COSNet

.2.1 Availability of source code and requirements

Project name: Par-COSNet
Project home page: `https://github.com/AnacletoLAB/ParCOSNet`
GitHub repository: `https://github.com/AnacletoLAB/ParCOSNet`
Operating system(s): Linux
Programming language: C++ with NVidia CUDA extensions library
Requirements: x86-64 processor, 256 MB RAM (dependent on graph size), NVidia GPU card with minimum Compute Capability 3.5 (v5.0 minimum for Dynamic Parallelism kernel); Compiled with GCC C++ compiler v5.4.0 of greater, NVidia CUDA Toolkit v8.0 or greater
License: Apache 2.0

.3 MCMC-Colorer

.3.1 Availability of source code and requirements

Project name: MCMC-Colorer
Project home page: `https://github.com/phuselab/MCMC_Colorer`
GitHub repository: `https://github.com/phuselab/MCMC_Colorer`
Operating system(s): Linux
Programming language: C++ with NVidia CUDA extensions library
Requirements: x86-64 processor, 256 MB RAM (dependent on graph size), NVidia GPU card with minimum Compute Capability 3.5; Compiled with GCC C++ compiler v5.4.0 of greater, NVidia CUDA Toolkit v8.0 or greater
License: MIT license