NeuroSuites-BNs: An open web framework for massive Bayesian networks focused on neuroscience

Mario Michiels, Pedro Larrañaga, Concha Bielza

Abstract

NeuroSuites-BNs is the first web framework for learning, visualizing, and interpreting Bayesian networks (BNs) that can scale to tens of thousands of nodes while providing fast and friendly user experience. All the necessary features that enable this are reviewed in this paper; these features include scalability, extensibility, interoperability, ease of use, and interpretability. Scalability is the key factor in learning and processing massive networks within reasonable time; for a maintainable software open to new functionalities, extensibility and interoperability are necessary. Ease of use and interpretability are fundamental aspects of model interpretation, fairly similar to the case of the recent explainable artificial intelligence trend. We present the capabilities of our proposed framework by highlighting a real example of a BN learned from genomic data obtained from Allen Institute for Brain Science. The extensibility properties of the software are also demonstrated with the help of our BN-based probabilistic clustering implementation, together with another genomic-data example.

Keywords: Bayesian networks, web, software, interpretability, neuroscience, genomics

1 Introduction

Currently, we are in the data era and have a wide range of approaches to analyse data ranging from statistical methods to machine learning techniques. The need for comparatively more advanced analysis techniques is rapidly increasing owing to the large quantity and complexity of data being generated.

This data explosion is occurring in almost each field because new datasets are being made available by new data acquisition technologies and data sharing platforms. In this study, we focus on neuroscience, which has always suffered from replication crisis caused by multiple small research teams not sharing their datasets and the use of ad hoc lab procedures. This scenario occurs despite numerous scientists in this field desiring access to public datasets of other researchers (Tenopir et al., 2011). However, this trend is changing because daily, more data are becoming available, and current collaborative frameworks are not only convenient but also necessary (Bouchard et al., 2016; Leitner et al., 2016; Wicherts et al., 2011).

Genomics and morphological/electrophysiological studies are particularly experiencing a tremendous rise in data. The contributions via collective databases are particularly important (e.g. NeuroMorpho.org (Ascoli et al., 2017) in the case of morphological data). In addition, some online collaborative frameworks such as brainlife.io (Hayashi and Pestilli, 2017) (mainly used for MRI studies) and Geppetto (Cantarelli et al., 2018) (which works with morpho-electrical biophysical models) are emerging to satisfy the need for processing and analysing all such datasets in an easy and useful man-

ner.

Analysing neuroscience data can be particularly complex as the data be obtained from multiple research areas with different scales and measurement tools. Some of these datasets can have numerous instances and/or present an extremely high dimensionality, such as microarray data, which can have a variable for every gene (in the order of tens of thousands). Learning machine learning models with massive datasets having numerous features requires unique algorithms, because they can cause the curse of dimensionality problem.

Besides the complexity of learning, obtaining the models and analysing them are key aspects to acquire useful insights about a domain. In biological domains, it is particularly sensitive and risky to make decisions based on models for which the process of drawing conclusions and their implications is not understandable. Furthermore, currently, it is particularly important to comply with the right to an explanation from algorithmic decisions (Goodman and Flaxman, 2017) to ensure they can be easily understood by the experts in the domain.

We need to differentiate between the trends of explainable artificial intelligence (XAI) (Nott, 2017) and interpretable AI. XAI focuses on modifications of well-established black box models, such as deep neural networks and random forests, to allow extracting knowledge from them, as they can provide accurate predictions in some domains but it is lacking in the explainability requirements (Rudin, 2019). Contrastingly, interpretable AI refers to white box models that are inherently explainable, which is the focus of this paper. Therefore, the current trend (Gunning, 2017) is directing to either conduct research on interpretable AI or XAI.

In the following, we refer to the explanatory capabilities of interpretable AI models as interpretability. To better understand the interpretability requirements, these models must have two properties: (a) conclusion transparency: it should be possible to ascertain why a conclusion was reached, e.g. by showing the obtained relationships between the variables, (b) improvement of the system: the model transparency should allow not only to validate the results but also to change the model when desired and validate how these changes affect the conclusion. For a more detailed view, we refer to Asilomar AI Principles (2017) and Samek et al. (2017).

We focus on probabilistic graphical models, particularly on Bayesian networks (BNs) (Pearl, 1988), because they fulfil all the above-mentioned interpretability requirements. The graphical component in a BN is particularly useful because it presents the probabilistic relationships between the variables. In addition, the inference machinery offers prediction and interpretability capabilities about the reasoning and the model. For a more in-depth review of the interpretability features of BNs, we refer the reader to Lacave et al. (2007) and Yuan et al. (2011). Owing to their interpretable nature, BNs have already been applied to neuroscience data with successful results (Bielza and Larrañaga, 2014; Luengo-Sanchez et al., 2019).

BNs are probabilistic graphical models that use the probability theory to present a compact graphical representation of the joint probability distribution over a set of random variables, $\mathcal{X} = \{X_1, ..., X_n\}$. BNs consist of two main parts: a graph, which is a directed acyclic graph (DAG) representing the probabilistic conditional dependencies between the variables in $\mathcal{X}$, and parameters, which are a series of conditional probability distributions (CPDs) (Koller and Friedman, 2009).

Each node in the graph represents a random variable, X_i, in the vector of variables, $\boldsymbol{X} = (X_1, ..., X_n)$, and its arcs represent the probabilistic conditional dependence relationships with respect to the other variables. Each node, X_i, has an associated CPD, which represents its probability distribution conditioned on its

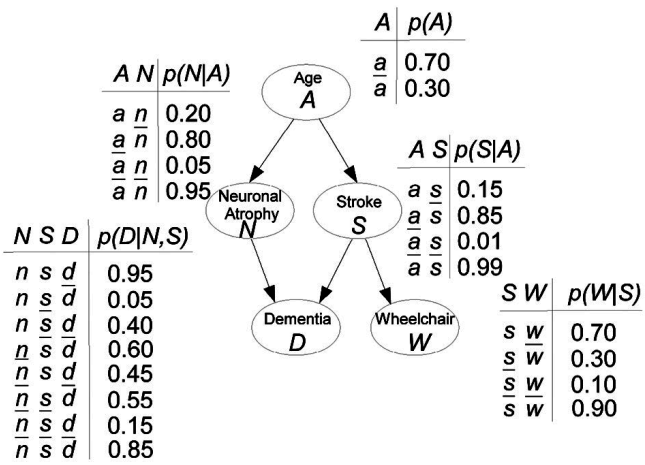


Fig. 1: **Hypothetical BN example modelling the risk of dementia.** Figure extracted from Bielza and Larrañaga (2014).

parents, $\mathbf{Pa}(X_i)$, in the graph (Figure 1). With all this information, the joint distribution of all the random variables can be expressed as

$$P(\boldsymbol{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)). \tag{1}$$

Following the example in Figure 1, if a patient has neuronal atrophy but has not had a stroke, using inference tools we can calculate that there is a 0.40 probability he will be demented: $P(d|n,\bar{s}) = 0.40$.

Although theoretically obtained properties are sufficient to explain a model, we need appropriate software tools to visualize and manipulate the model interactively. The remainder of this paper is structured as follows: first we review the requirements that BN software tools should meet to fully exploit their interpretability features: scalability, extensibility, interoperability, ease of use, and interpretability.

In the next section, we review the existing multiple software tools in the state-of-the-art and highlight that all of them lack one or more fundamental aspects so that they cannot visualize all the BN capabilities. Then, in Section 3, we present NeuroSuites-BNs, a new open-source framework that aims to address all these key factors and provide real-world use cases. Finally, we discuss future improvements to be implemented in this line of research.

2 Problems with state-of-the-art of software in massive BN interpretability

It is important to differentiate between individual software components addressing specific tasks (e.g. a learning algorithm), referred as software packages, and general frameworks, as the one presented in this paper, which provide all the necessary tools to work with BN capabilities (learning, visualization, and reasoning). In this section, we review the problems with the current BN software frameworks and packages by explaining the contents summarized in Table 1, which compares all the main BN software.

2.1 Scalability

Massive BNs present mainly three scalability problems: learning their graph structure, efficiently visualizing it, and developing a fast inference engine for the reasoning.

When the number of variables is extremely small, the graph structure of a BN can be even modelled with only expert knowledge. However, when the dataset has numerous random variables, the graph must be learned by computational methods. Learning the structure of a BN is known to be an NP-hard problem (Chickering et al., 1994). The search space for all the possible DAGs is super-exponential in the number of nodes, i.e. $O(n!2^{\binom{n}{2}})$ (Robinson, 1973). Different algorithms attempt to solve the above problem by applying heuristics to this super-exponential space.

The problem becomes comparatively more complex when dealing with a massive number of variables of the order of thousands of nodes, requiring distinct types of algorithms for constraining the computational memory and time. This problem can be solved in a reasonable time by two methods: constraining the graph structure and developing new algorithms that completely uti-

lize parallelization technologies. The first solution includes algorithms that inherently constraint the structure (e.g. the Chow–Liu method (Chow and Liu, 1968)) and the generating poly-tree recovery algorithm (Rebane and Pearl, 1987); in the latter, the resulting graph can only be a tree or a polytree. There are other algorithms which by default do not constraint the structure; however, when the problem has an extremely high dimensionality, they include assumptions, like limiting the number of parents, for each node to finish in a reasonable time. Some examples of this case are the PC algorithm (Spirtes et al., 2000) and the max–min hill-climbing (MMHC) algorithm (Tsamardinos et al., 2006). For a more detailed view of BN structure learning algorithms, we refer the reader to Koski and Noble (2012).

However, some problems like learning gene regulatory networks (GRNs) need to be modelled without restricting the structure, because all types of relations between the variables are possible. The algorithms available for these models are highly limited because most of them cannot be parallelized; therefore, new optimized algorithms are emerging (Bernaola et al., 2019; Liu et al., 2016; Madsen et al., 2017). Another problem is that some of these state-of-the-art algorithms are not typically found in the existing BN software frameworks, because the latter are not frequently updated to include new algorithms. Moreover, the existing frameworks do not have a scalable software architecture to parallelize these algorithms on multiple computing nodes.

Once the BN structure and its corresponding parameters have been learned, we need approaches to visualize it efficiently to understand the model and to analyse it to draw conclusions. Although there exist software packages that can visualize general-purpose massive graphs (Graphistry; Jacomy and Plique; Kashcha) using the GPU computational power, this is not so for BNs. Specifically, for BNs, viewing the nodes and edges is not sufficient; we also need to visualize their node parameters and run BN operations, such as making queries and observing the posterior distributions. Essentially, we need a rich set of interactive options to fully understand and exploit the graph structure and parameters. This is clearly one of the most important bottlenecks in the current frameworks when dealing with massive BNs.

Finally, we require an efficient inference engine, which in the ideal case would be exact. However, exact inference in discrete BNs is NP-hard (Cooper, 1990); therefore, the network structure can be constrained to reduce this cost with unique algorithms. Approximate inference is the alternative when we do not want to constrain the network structure; however, it is also associated with computational problems as it is also NP-hard (Dagum and Luby, 1993). In comparison, exact inference is tractable in the continuous space for Gaussian BNs (see Section 3.4.4).

2.2 Extensibility

Extensibility refers to the software capability to include new functionalities easily and coherently. It is crucial for the software to be written modularly to introduce new algorithms and functionalities. Three of the major software in BN frameworks are BayesiaLab (Conrady and Jouffe, 2013), Hugin (Madsen et al., 2005), and BayesFusion (Druzdzel, 1999), which all have proprietary licenses, and therefore, the code is not open-source. This presents a significant problem in an advancing research field like this, because the research community cannot code its own extensions and changes. In an ideal case, the frameworks should be open-source and have simple and multiple approaches to introduce new extensions coded by the community.

2.3 Interoperability

Interoperability can be considered as one of the direct consequences of a good extensibility level. This is because it refers to the capability of integrating new func-

tionalities (from other software packages to improve their usability) and possessing an ecosystem where all the components can work together.

The current frameworks (Table 1) are proprietary and specifically designed only for working with probabilistic graphical models. Therefore, connections with other types of machine learning algorithms or different analysis tools are not possible at present. Owing to their proprietary nature, the community developers cannot implement some functionalities, such as having direct API connections with specific data sources as neuroscientific databases.

Nevertheless, the BN community has some open-source software packages that are well maintained and have a good extensibility; however, they are designed for highly specific tasks, e.g. learning algorithms (Aragam et al., 2019; Benjumeda et al., 2019) and inference algorithms (Højsgaard, 2012). We also have other packages, such as bnlearn (Scutari, 2010) and pgmpy (Ankan and Panda, 2015), which comprise a set of interconnected tools but they lack some basic modules, e.g., a graphical interface or connection with other packages, which would make them to be considered as frameworks. Thus, the problem of such packages is the lack of completeness, unlike the proprietary options.

Furthermore, some software packages are developed for the specific purpose of a scientific research. While this is appropriate for advancing the research field, frequently these software tools are overlooked and not maintained once the corresponding paper is published. The first consequence is a waste of time associated with coding again previously written algorithms by other researchers when the existing code becomes obsolete and not extensible. Another consequence is the difficulty of integration to other software, because they may be written in a different programming language. Therefore, the library can have data format problems, software incompatibilities between versions, etc.

2.4 Ease of use and interpretability

Software packages regularly do not include a graphical interface; therefore, the learning curve is extremely steep for users not experts in programming, which commonly is the case with some neuroscientists. Graph visualization cannot even be considered for software packages because they mostly rely on third-party software to display a static view of the BN structure and cannot display node parameters.

In comparison, frameworks are much more user friendly because they provide a sophisticated graphical interface to work with. However, as a direct implication of their low scalability, they are not capable of visualizing and managing massive networks. Moreover, specific solutions for distinct use cases (e.g. automatically running a set of algorithms when new data emerges from a database) cannot be developed by different research teams, because of their extensibility problem. This problem is another bottleneck when customized solutions need to have an easy and rapid workflow, ranging from acquiring the data to analysing them.

3 NeuroSuites-BNs

In this section, we present NeuroSuites-BNs, whose software architecture has been specifically designed to overcome all the problems highlighted in the previous section (see also Table 1). Summarizing, we find ourselves stuck in incomplete open source solutions versus more complete solutions in the proprietary field. The objective of the framework presented in this paper is to combine the best properties of both worlds and present one complete open-source solution, with the possibility of further improvement and becoming increasingly complete. It is important to acknowledge that NeuroSuites-BNs is included in the NeuroSuites platform, which we developed for integrating different neuroscience tools.

NeuroSuites-BNs has already been successfully

used with genomic data in Bernaola et al. (2019), and as genomic examples, here we present real-world use cases to illustrate how we addressed the four interpretability requirements explained above.

Tab. 1: Comparison of the main BN software frameworks/packages

	Features/software	NeuroSuites-BNs	BayesiaLab	BayesFusion	Hugin	bnlearn	pgmpy
Scalability	Learn massive networks	✓					
	Visualize massive networks	✓					
	Parallelized learning (single computer)	✓			✓	✓	
	Parallelized learning (cluster computing)	✓				✓	
	Has web interface	✓	✓	✓			
Extensibility	Open source	✓				✓	✓
	Discrete variables learning	✓	✓	✓	✓	✓	✓
	Discrete variables inference		✓	✓	✓	✓	✓
	Discrete variables visualization	✓	✓	✓	✓		
	Continuous variables learning	✓		✓	✓	✓	
	Continuous variables inference	✓		✓	✓	✓	
	Continuous variables visualization	✓		✓	✓		
	Probabilistic clustering	✓	✓				
	Dynamic BNs		✓	✓	✓		✓
Interoperability	Connection with other languages	✓		✓	✓		
	Connection with other science fields	✓					
	Connection with online data sources	✓	✓	✓			
	Import/export BNs from/to other software	✓	✓	✓		✓	✓
ease of use	Is a framework	✓	✓	✓	✓		
	Interactive visualization	✓	✓	✓	✓		
	Interpretation of massive networks	✓					
	Available online	✓					

3.1 Scalability

NeuroSuites is developed as a scalable web application to run the heavy operations in the backend while providing a lightweight rapid experience in the frontend. Its framework follows a modular architecture (Figure 2), where each fundamental component is isolated as a Docker (Merkel, 2014) microservice (Figure 2.1); therefore, the system can be easily scalable horizontally and work as a whole. Moreover, multiple monitoring tools have been included since the architecture became large and complex, and a set of tools is provided to monitor the state of the hardware, logs, task queues, etc.

The scalable architecture is designed to be efficient and solve the computational problems of visualizing and managing large learning algorithms and graph operations. The nginx web server (Sysoev, 2004) provides the entry point for the web requests (Figure 2.2) and also acts as the load balancer in case the server has multiple instances running the web application.

The frontend code (Figure 2.3) is based on vanilla JavaScript (JS) and JQuery, to provide a simple but efficient architecture, and the style is in HTML5, CSS3, and Bootstrap3. To provide a scalable visualization of the BN graphs, we have made various extensions to the sigmajs library (Jacomy and Plique), which range from visualizing the node parameters to numerous specific BN operations, fully explained in Section 3.4. Sigmajs uses a WebGL engine, which utilizes a GPU card to efficiently visualize massive graphs.

To transmit the requests and responses from the frontend to the backend, we employ the uWSGI software (Unbit), which acts as a web server gateway interface to communicate with the Python backend (Figure 2.4). The backend core (Figure 2.5) is written in the Django framework (Django Software Foundation, 2013), to allow us to use optimized Python libraries for managing the graph and also other scientific libraries (e.g. Numpy, Scipy, or Scikit-learn (Jones et al.; Pedregosa et al., 2011; Van Der Walt et al., 2011)). NetworkX (Hagberg et al., 2008) is the main library used in the backend to store the graphs and run the graph manipulation tasks. Lightweight graph operations, such as colouring groups of the nodes, are completely conducted in the frontend with the sigmajs library. The heavyweight operations are sent to the backend where they are processed with NetworkX, and the result is sent back to sigmajs to update the graph (Figure 2.6).

Standard HTTP requests and responses have time and computational limitations, which make them unfeasible to run long-duration tasks, e.g. some BN structure learning algorithms. To overcome these limitations, we have included a queue-workers system using RabbitMQ (RabbitMQ) and Celery (Celery) (Figure 2.7). The system arranges all the long time-consuming requests and queues them to be executed in the most efficient order. The system administrator can opt to scale the necessary containers when the workload is not sufficient for the number of concurrent users. For instance, when the workload is highly intense in the heavy operations, the system administrators will increase the number of workers and the queue system will automatically distribute the workload.

For high memory efficiency, the uploaded datasets are internally stored on our server using the Apache Parquet (Vohra, 2016) format . To save the internal state of an application, the data session of the user is stored in a PostgreSQL database (PostgreSQL) connected to the Django framework to process all the operations in transparently (Figure 2.8).

The included BN structure learning algorithms are categorized into the following six groups: (a) Statistical based (from Scikit-learn (Pedregosa et al., 2011), only for continuous variables): Pearson correlation, mutual information, linear regression, graphical lasso, and GENIE3 (Irrthum et al., 2010); (b) Constraint based: PC, grow shrink, iamb, fast.iamb, and inter.iamb; (c) Score and search: hill climbing, hill climbing with tabu search,

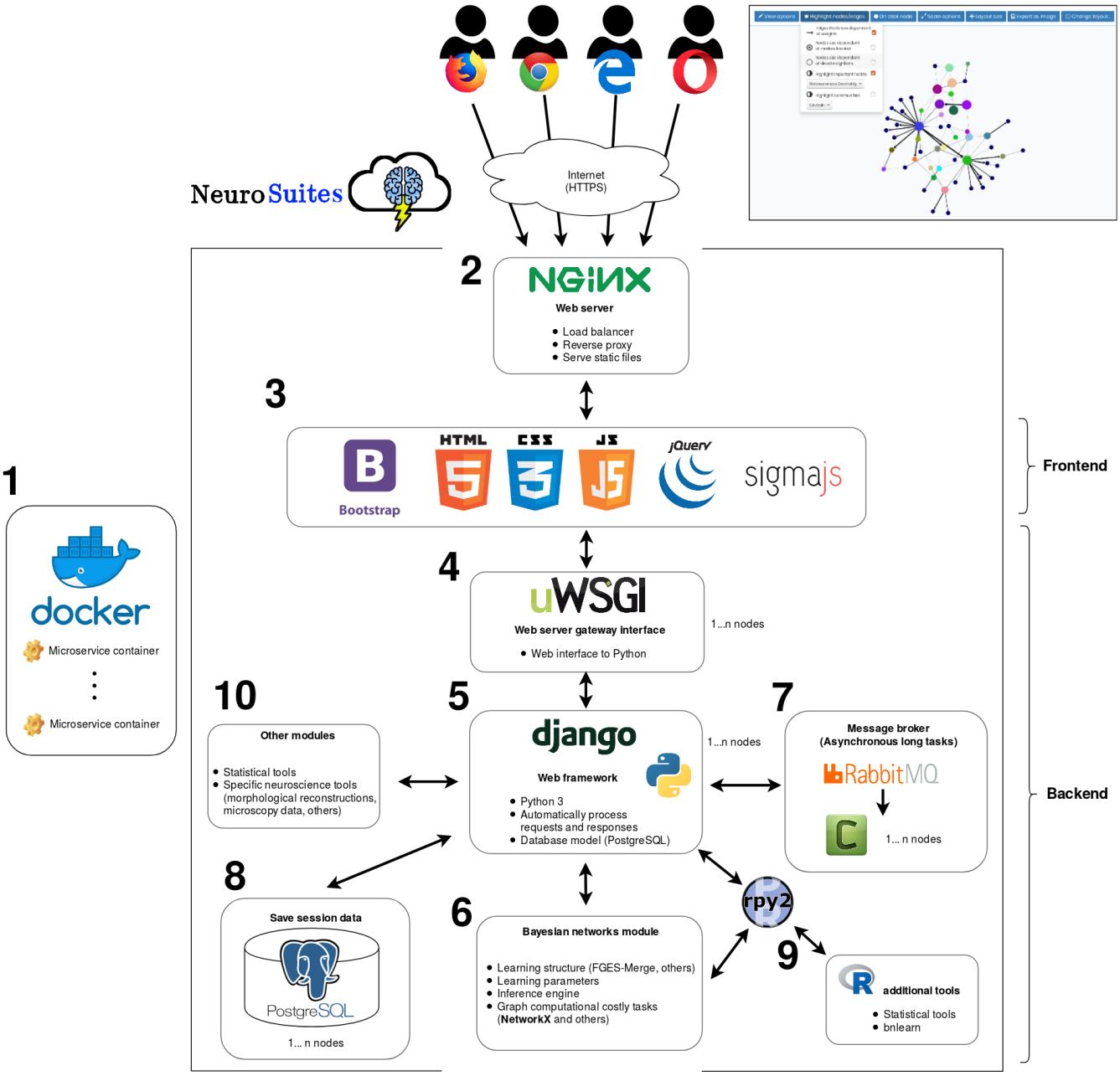


Fig. 2: **Software architecture of NeuroSuites-BNs.**

Chow-Liu tree, Hiton parents and children, sparsebn (Aragam et al., 2019), and FGES-Merge (Bernaola et al., 2019); (d) Hybrid: MMHC and MMPC; (e) Tree structure: naive Bayes, tree augmented naive Bayes; (f) Multi-dimensional Bayesian network classifier. All the algorithms where we have not specified a reference here, were implemented in bnlearn.

Only some structure learning algorithms are suitable for large-scale networks, such as the Chow–Liu algorithm, GENIE3, sparsebn, MMHC, and FGES-Merge. However, for massive networks only the FGES-Merge can learn a network in a reasonable time without constraining the structure, because it is coded to run in parallel in multiple computing instances. NeuroSuites-BNs includes MPI (Walker and Dongarra, 1996), which allows this type of parallelism using the mpi4py Python package (Dalcin). However, owing to the computational limitations of our production server, we cannot provide more than one computing node. Nevertheless, developers who install their own NeuroSuites instance can

make use of this parallelization capability by deploying multiple computing nodes to run their desired Docker containers.

BN parameter learning and inference engine (Figure 2.8) have also been designed to be scalable for massive BNs and are explained in detail in Sections 3.4 and 3.4.4, respectively.

3.2 Extensibility

NeuroSuites follows an extensible architecture where each module has internal submodules, allowing the modules to extend in any manner.

This extensibility enables highly easy integration of new learning algorithms or new graph functionalities for BNs and other modules. For example, to include a new structure learning algorithm, the only requirements are taking the dataset as a Pandas data frame (McKinney, 2010) and outputting a NetworkX graph object. The changes in the frontend would be minimal, only adding a new button to run the new learning algorithm. The entire workflow is automated, and the learning algorithm would be directly queued to the system when a request is sent.

As the backend core is written in Python, the easiest method to extend it is by coding a Python extension. Because we aimed to support maximal scientific communities, we also included bindings to the R programming language for the BN learning algorithms and other statistical packages. The binding was easily achieved via the wrappers provided using the Rpy2 package (Gautier) (Figure 2.9).

To demonstrate the extensibility of the models, we also included support for BNs-based clustering models. Thus, in the backend side, a subclass of the BN model was created with the necessary extensions, and for the frontend side, the same Javascript core for BNs was recycled and the necessary extensions were included (see Section 3.4.4).

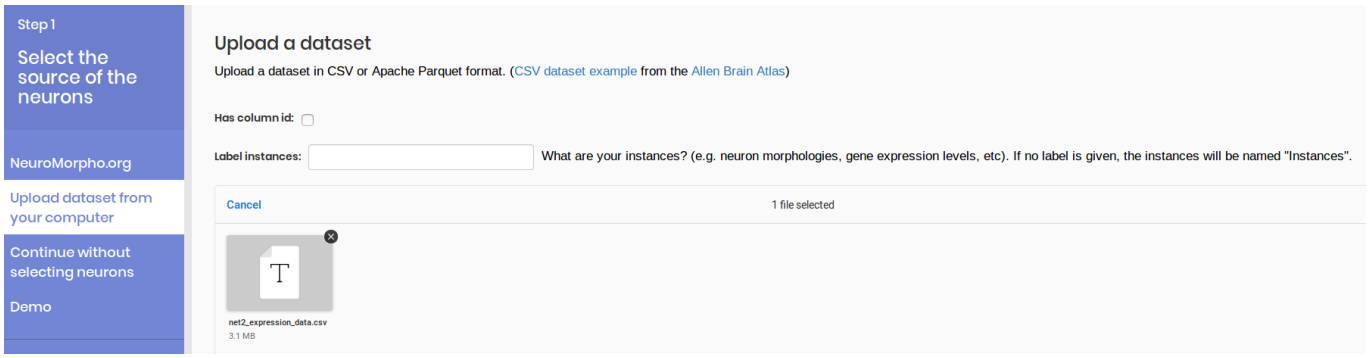
3.3 Interoperability

To provide an interoperable ecosystem, we designed a well-defined workflow consisting of first uploading the raw dataset and then selecting the desired tools to analyse it. Therefore, different sections can be found on NeuroSuites, where each refers to a tool or a specific set of processing tools. The focus of this study is on the BNs section; however, users can also use other tools already integrated in NeuroSuites. Some of these tools, such as the statistical analysis section (Figure 2.10), can provide significant preliminary analysis for improved better understanding of the data to then create better BN models.

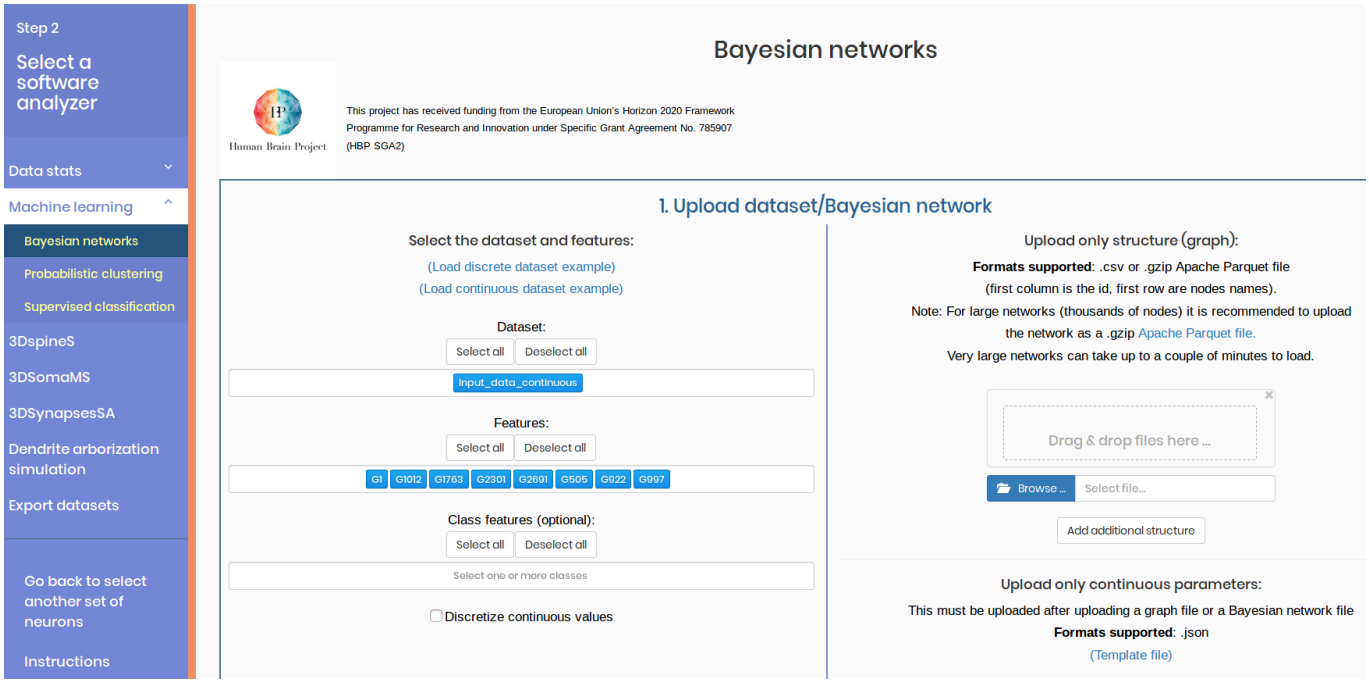
As a use case regarding interoperability, there exists an API client that can connect a data source; it is the latest live version of the NeuroMorpho.org database. This type of direct connection to a data source is convenient when the data from a specific neuroscience field are required to be connected. This allows the users to easily access the data without the need to first download the data on their computer and then upload them to the NeuroSuites server. Thanks to the extensibility properties of NeuroSuites, it would be straightforward to implement numerous data source connectors to any database, e.g. the Allen Cell Types Database (Sunkin et al., 2012) and the DisGeNET database for genes-human disease associations (Piñero et al., 2016).

3.4 Ease of use and interpretability

Here, we review the capabilities of NeuroSuites-BNs by presenting a complete real use case: learning and interpreting the GRN of the full human genome using brain data extracted from microarrays, provided by the Allen Brain Atlas (Hawrylycz et al., 2012). The dataset consists of 20,708 protein-coding genes as predictor features with 3500 samples; therefore, each element in the dataset corresponds to a measurement of a gene expression level.



(a) Upload data set section



(b) Select variables to create the BN

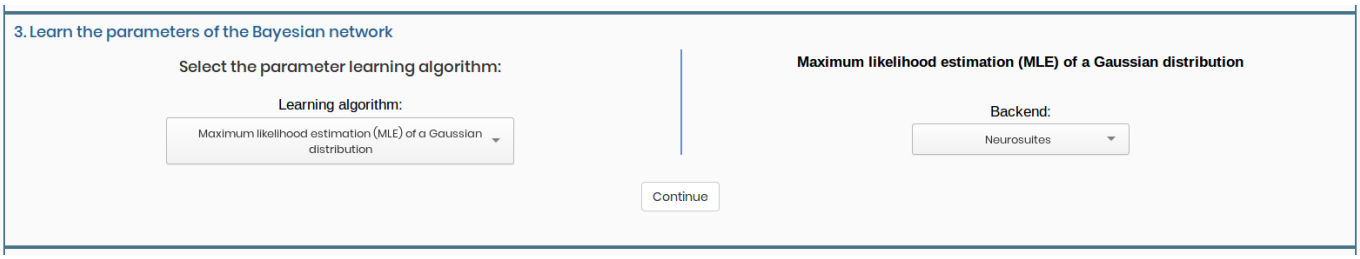
Fig. 3: **Steps to upload a data set and select its desired variables.**

In step 1, the desired dataset (Figure 3a) is uploaded. In our deployed production server, we accept CSV and Apache Parquet gzip formats. Note that the BNs can also be created by different software, e.g. BayesiaLab or bnlearn, and then be imported in a BIF/CSV/Apache Parquet format to NeuroSuites-BNs to visualize and interpret the model. However, for this example, we present the entire workflow to create and interpret a new model.

In step 2, we move to the BNs section under "Machine Learning" and select the desired variables to learn the model (Figure 3b). For this example, we select some continuous variables that correspond to the expression level of some genes. It is also possible to discretize the selected variables with different methods or select the class variables for a supervised classification model; however, this is not the case in our example.

Following the selection of the desired variables, the BN structure graph is learned by selecting a structure learning algorithm, as described in the section below (Figure 4a). For this example, we use FGES-Merge because it is specifically designed for genomic data, being memory and computationally efficient and having the ability to readjust the final structure to follow the

(a) BN structure learning algorithm selection



(b) BN parameter learning algorithm selection

Fig. 4: **Steps to learn a BN.**

topology of the GRN (Nair et al., 2015).

Once the algorithm is completed, the obtained graph is displayed in the visualizer, and we can immediately manipulate it. Nevertheless, to provide a complete example, we also present how to learn the model parameters for each node. For this, we select the maximum likelihood estimation (MLE) of a Gaussian distribution (Figure 4b), which provide the learned Gaussian distribution for each node and the weights corresponding to the relationships with its parents. Mathematically, the CPD of a node, Y, given its parents $\mathbf{Pa}(Y)$ is

$$p\left(Y|\mathbf{Pa}(Y)\right) = \mathcal{N}\left(\beta_0 + \boldsymbol{\beta}^T\mathbf{Pa}(Y); \sigma_Y^2\right). \qquad (2)$$

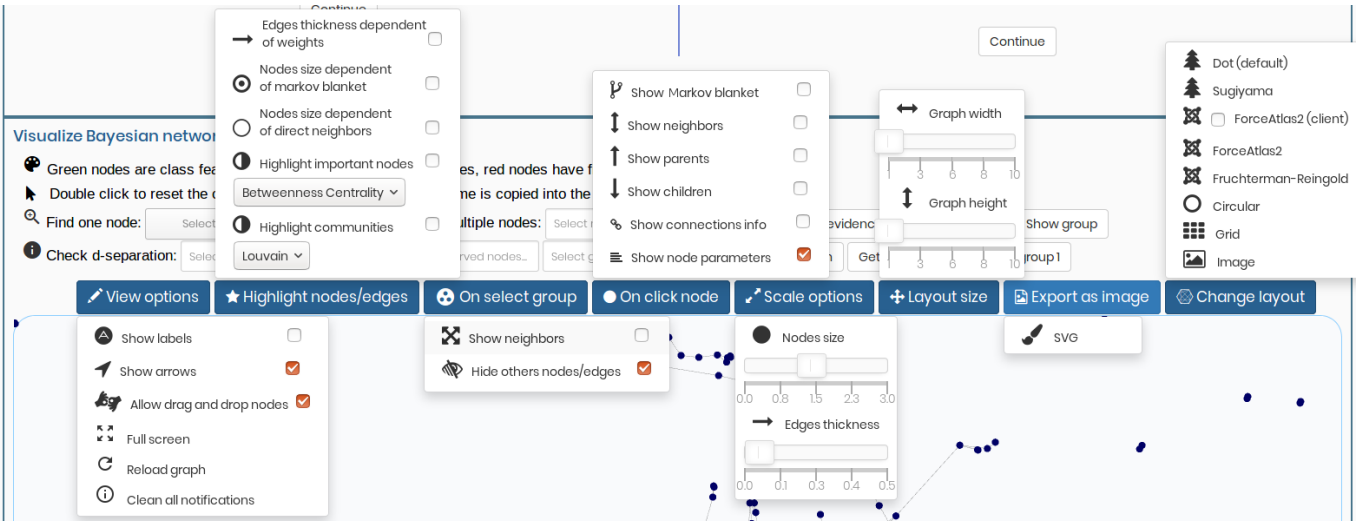
To estimate the parameters, β_0, $\boldsymbol{\beta}$, and σ_Y^2, for each node, the Gaussian MLE learns a multilinear regression between Y and $\mathbf{Pa}(Y)$. The regression coefficients provide estimations of β_0 and $\boldsymbol{\beta}$, and the mean of the regression residuals sum of the squares yields the σ_Y^2 estimate.

Having learned the node parameters, we can utilize the inference engine by asking some queries to the BN and obtain the predicted results when some node values are fixed, as explained in detail in Section 3.4.4.

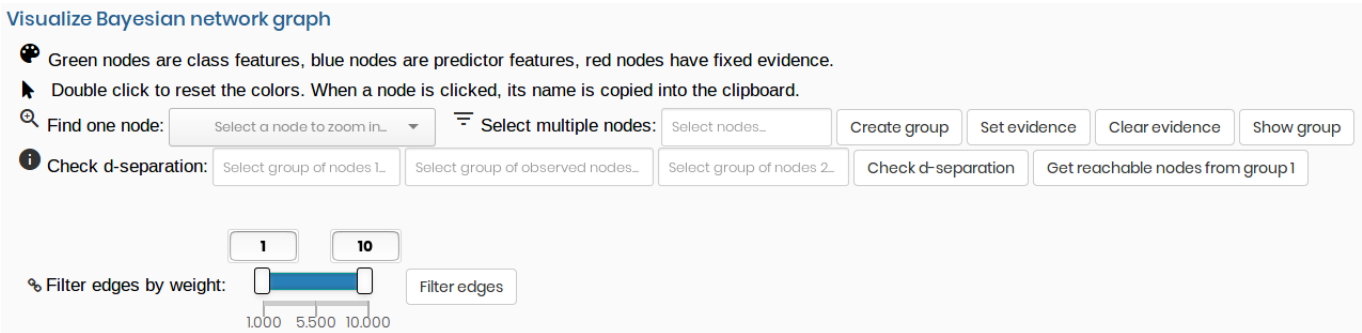
There are several visualization and interpretability options, which are categorized into four groups: layouts, general viewing options, highlighting nodes/edges, and parameter visualization and inference.

3.4.1 Layouts

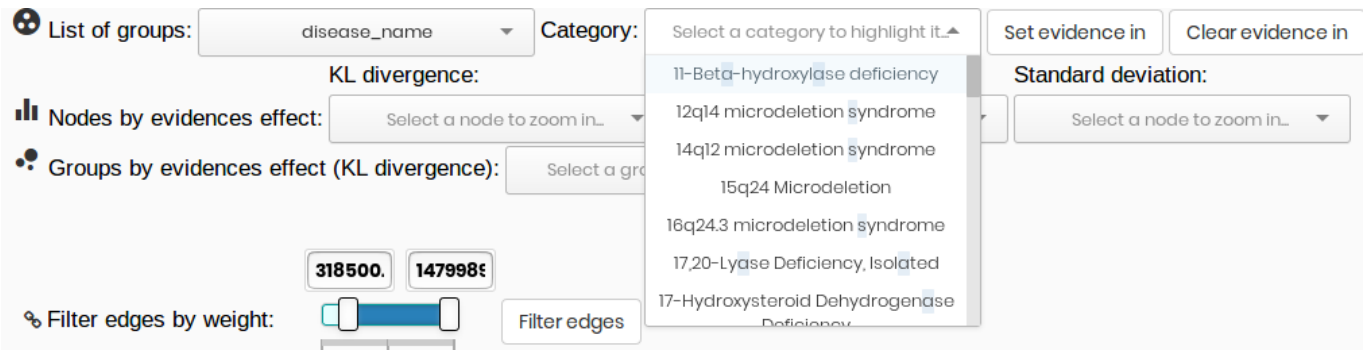
A predefined layout is displayed in the visualizer when the BN is loaded for the first time, but depending on the problem, a different one might be needed to be set. Choosing the appropriate layout should be the first step to understand the graph structure of a BN. The layouts (Figure 5a, right corner) can be tree-based layouts (Dot, Sugiyama) (Koutsofios and North, 1991; Sugiyama et al., 1981), force-directed layouts (Fruchterman-Reingold, ForceAtlas2) (Chippada; Csardi and Nepusz, 2006; Fruchterman and Reingold, 1991; Jacomy et al., 2014), circular, grid, and image



(a) Lower bar options



(b) Upper bar options



(c) Lower bar options when groups are created and inference is conducted

Fig. 5: **BN visualization options.**

layouts. The last one is a novel method developed by us to create a layout by detecting the edges from any image. It is particularly useful for creating user-defined layouts or complex layouts that cannot be implemented by other algorithms. Layouts are computed in the back-end side for efficiency, although we also provide a front-end (client version) implementation for the ForceAtlas2 algorithm (Plique, 2017).

For small or medium BNs, tree layouts are recommended, whereas force-directed layouts are recommended for large BNs, because with this type of layout cluster formation occurs. In this example, we select the

ForceAtlas2 algorithm because it can clearly yield the topology properties of GRNs (locally dense but globally sparse) (Figure 6a). Note that the extensibility nature of a project affect the convenience for the developers to add new layout algorithms or modify the existing ones to meet their own needs.

3.4.2 General viewing options

For general viewing options, we can easily navigate through the graph, allowing to zoom any region of interest. The lower bar of the visualizer has buttons to show/hide the labels for each node, arrows, drag and drop nodes, full screen, and reloading the graph (Figure 5a, left side).

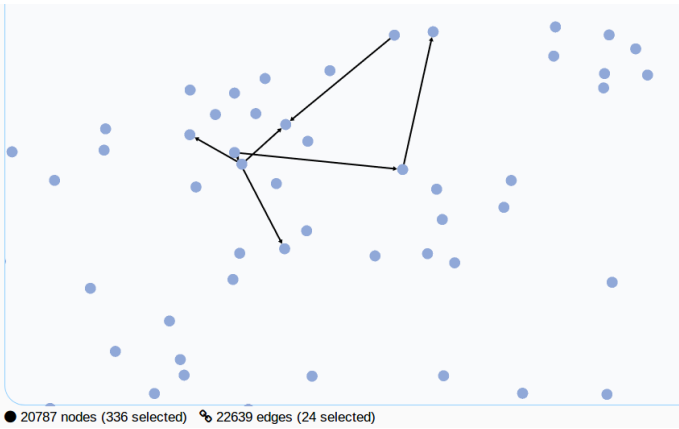
Multiple relevant scale options are also implemented (Figure 5a, right side), such as node sizes dependent on the number of nodes in their Markov blanket or edge thickness dependent on their weights, irrespective of their reference. For instance, the edge weights can correspond to a score that refers to their importance in the network, such as the BIC score (Schwarz, 1978). It is a penalized likelihood of the dataset calculated with the BIC difference of adding that edge versus not adding it. A filtering option to remove the edges below or above a certain weight threshold is also included (Figure 5b, bottom left).

3.4.3 Highlighting nodes/edges

Subsequent to selecting the appropriate layout and configuring the general viewing options, the next step is highlighting the relevant nodes or edges. We provide tools for highlighting the nodes isolated in the Markov blanket of a given node or its parents or children (Figure 5a, centre).

When dealing with massive networks, one of the most important features is the creation of groups. The groups can be created by three ways: manually, automatically, or uploading a list of already defined groups

(a) Nodes coloured by the Louvain algorithm for communities detection. ForceAtlas2 layout is applied



(b) Metadata groups information uploaded. Same network as in (a) but now only a subset of the nodes associated with the schizophrenia disease and the edges between them are selected

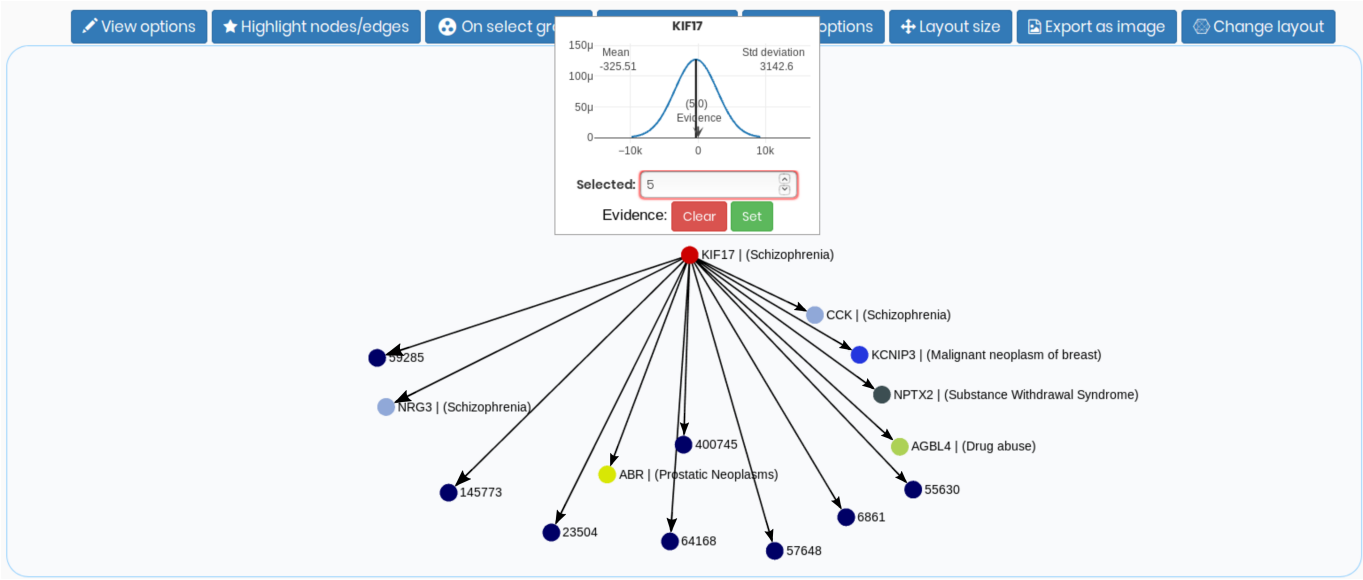
Fig. 6: **BN structure of the full human brain genome, where independent nodes are not shown.**

of nodes. A node or a set of nodes manually can be selected by searching for them by their name in the search fields with auto-completion (Figure 5b, middle left, ”Find one node”). Once we have selected the desired nodes to highlight, we can opt to create a group with them, and our node selection is saved to be used

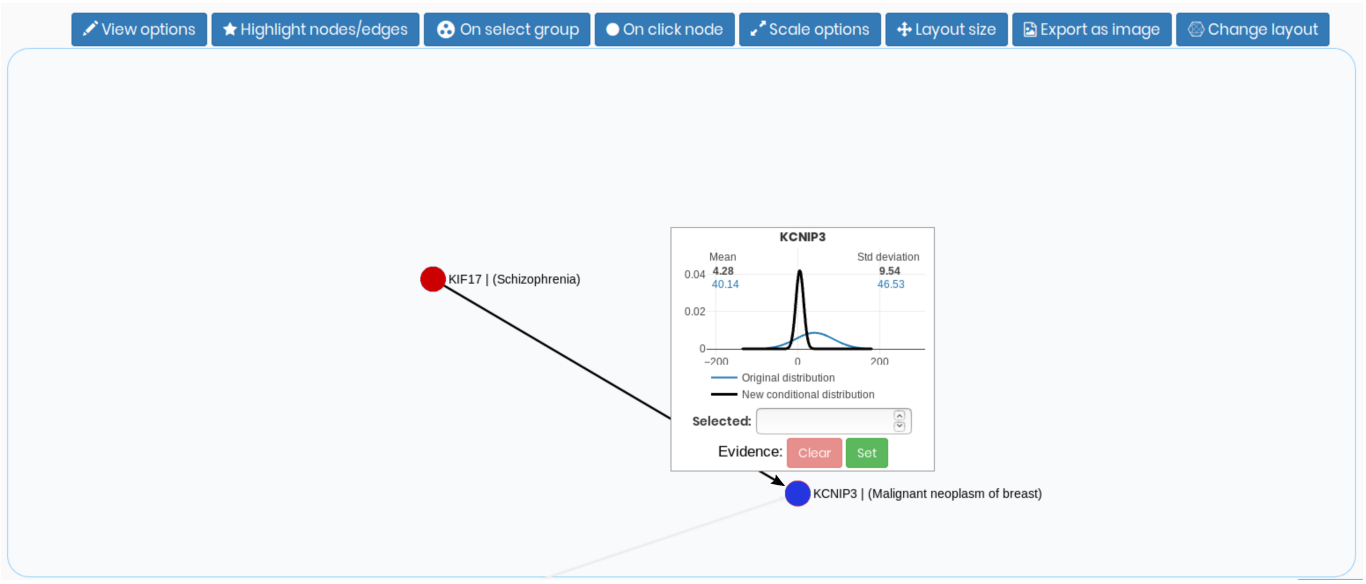
subsequently (Figure 5b, upper middle, ”Select multiple nodes”). A name and colour can also be assigned to each created custom group.

To generate groups automatically, we can run some algorithms designed for community detection, such as the Louvain algorithm (Blondel et al., 2008), which optimizes a modularity score for each community. In this case, the modularity evaluates the density of edges inside a community compared to that of the edges outside it. To select groups already created externally, we can upload the metadata JSON file, so that each node has some associated tags.

Finally, we can select a specific group (Figure 5c, upper left), and each node is displayed according to the colour of its category (Figure 6a). Moreover, we can select a specific category within a group (Figure 5c, centre), and only the nodes with that category are shown (Figure 6b).



(a) Selection of one random node associated with schizophrenia disease, following inclusion of the metadata information about the gene–disease association. In this case, we select the node on top, corresponding to gene KIF17, fix its value to make it an evidence node, $E = e$, and only show its children to have a clear view of their relations



(b) Evidence node set (KIF17) (top) and one of its children (bottom), corresponding to gene KCNIP3 associated with malignant neoplasm of a breast. The plot includes its original marginal Gaussian PDF in blue, $p(Q)$, as it is before setting any evidence, and the new one in black, $p(Q|E)$, which corresponds to its PDF after setting the evidence of gene KIF17. The exact parameters are also displayed. Therefore, the inference process demonstrates how fixing a low value for the gene associated with schizophrenia (KIF17) also results in a value near zero for the gene associated to the malignant breast neoplasm (KCNIP3), which indicates a relationship between these two genes.

Fig. 7: **Inference workflow in BNs.** The network corresponds to the full human brain genome from the Allen Institute for Brain Science.

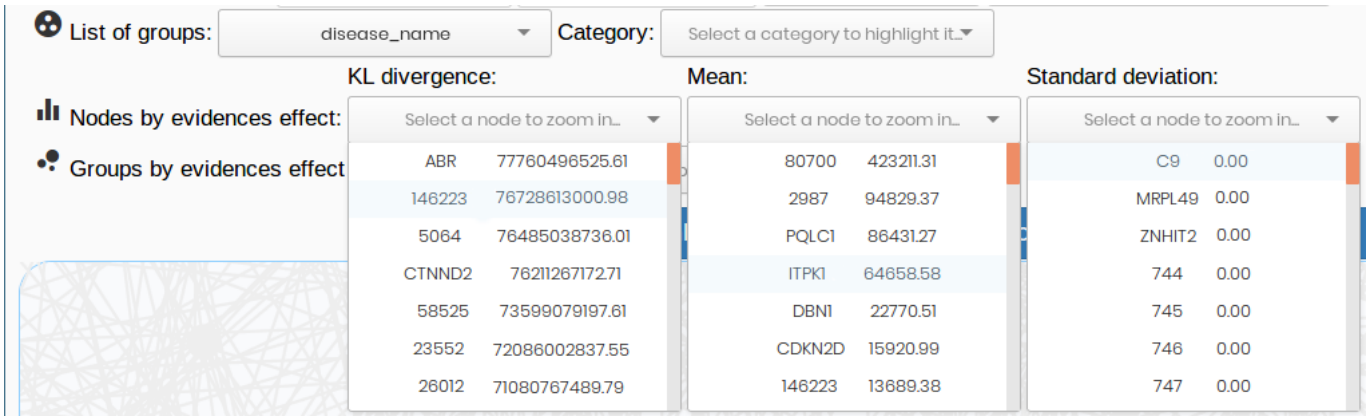


Fig. 8: **Inference effect in the query nodes.** We can now infer the extent the evidence of a node (or group of nodes) affects the PDF of other nodes or group of nodes, $p(Q|E)$, by examining the Kullback–Leibler (KL) divergence between the original and the posterior distributions or their mean or standard deviation variation. The left column in each drop-down box corresponds to the genes id, and the right column presents the score values. Note that in this example, the standard deviation values seem to be zero, because they are rounded to two decimals. Further, the effect of fixing the evidence of only one node in a network of more than 20,000 nodes can be minimal for the standard deviation of the other nodes.

When selecting a group of nodes, the arcs between these nodes are also selected to provide a clear view of the group. A user can also opt to highlight the neighbours of the nodes for that group, even if they do not belong to that group (Figure 5a, centre). Finally, to realize a clear understanding of where a group is within the global network, a user can enable an almost transparent view of all the other nodes that are not in the selected group.

Additionally, individual important nodes can also be selected by fixing a threshold for their minimum number of neighbours. An automatic approach has also been included to highlight the important nodes using the betweenness centrality algorithm (Figure 9a) implementation in NetworkX. It can detect the importance of a node is according to the number of shortest paths (for every pair of nodes) that pass through the node.

Comparisons of two different BNs are also possible by displaying both structures in the same graph and colouring the edges depending on which network they belong to. To achieve this, we must first upload a BN or learn it from a dataset, and then repeat this with the second BN. However, a visual comparison is not sufficient when the networks are large. Hence, we include a summary table displaying some structural measures, such as the accuracy, precision, recall, F-Score, and Matthews correlation coefficient, which use the confusion matrix of the edge differences of the second BN with respect to the first BN.

3.4.4 Parameter visualization and inference

The next step is to visualize the node parameters and make some queries to the BN, to demonstrate how the inference engine works. NeuroSuites-BNs supports visualization for both discrete and continuous nodes. In the case of discrete nodes, the marginal CPD table is provided, whereas in the continuous case, an interactive plot of its marginal Gaussian PDF is displayed (Figure 7a).

Because our example has only continuous Gaussian nodes, we describe the continuous exact inference engine. This involves converting the network parameters into a multivariate Gaussian distribution, $\mathcal{N}(\boldsymbol{\mu}; \boldsymbol{\Sigma})$;

therefore, the marginalization operation for a query variable, $p(Q = q)$, is trivial, because we only need to extract its mean and variance from the multivariate Gaussian distribution. For the conditioning probability density of a query variable given a set of evidence variables, $p(Q = q|\boldsymbol{E})$, we also utilize the multivariate Gaussian distribution, following the equations described in Koller and Friedman (2009).

Performing the inference operations in this manner allows a highly rapid inference engine because the most time consuming operation is conditioning over a large set of evidence variables in $\boldsymbol{E}$, which is $O(l^3)$, being l is the number of evidence variables $\boldsymbol{E}$ to condition on. This complexity is directly a result of the formulas for conditioning, as it is needed to invert a matrix of size $l \times l$.

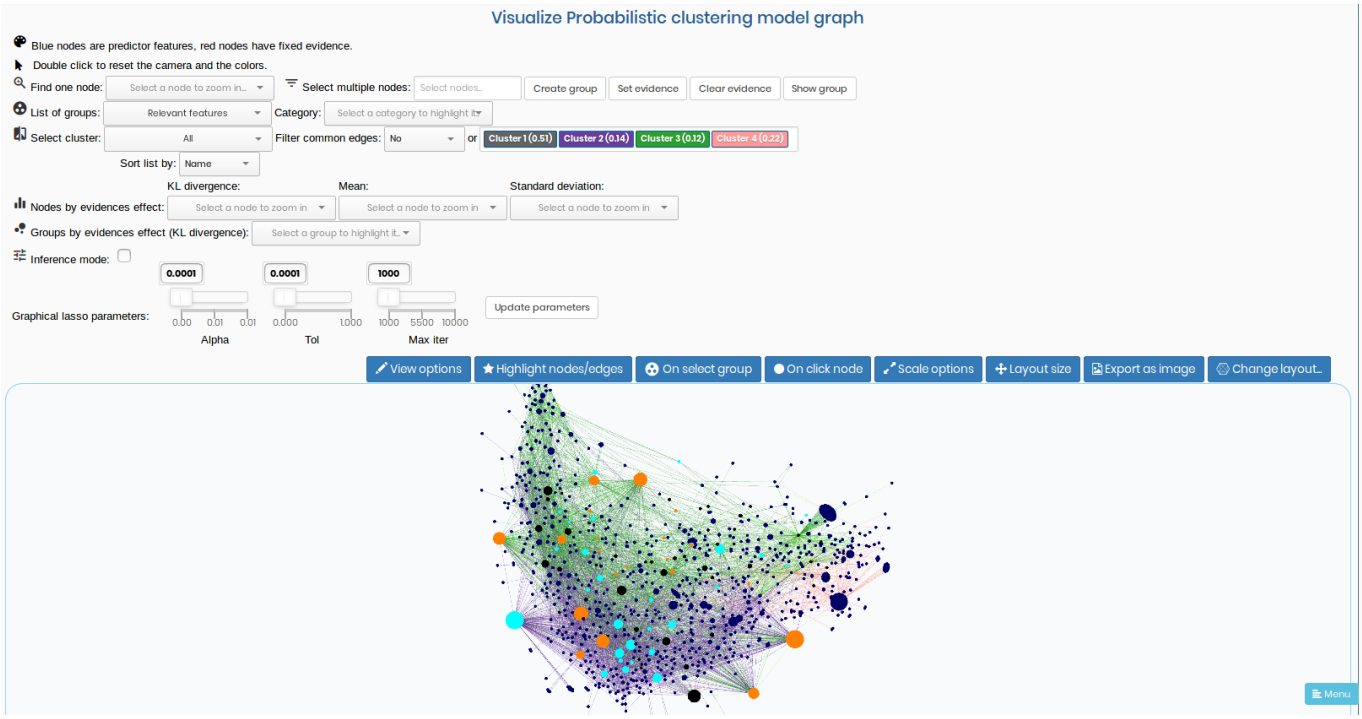
From the user perspective, this entire process is transparent, which is a key factor for the ease of use and interpretability of BNs. The inference process is as follows: to set the evidence nodes, $\boldsymbol{E}$, the user either clicks on the desired node and fixes the exact value (Figure 7a) or selects a group of nodes. The last option only allows fixing a shared value of the evidence for the whole group, because the standard deviation of each member of the group varies from its mean value. Setting different values at each node would be inefficient because the group can be large and the nodes can have different scales.

To view how the parameters of the query nodes, $p(Q = q|\boldsymbol{E})$, change, the user clicks on a specific node and both the original and new distributions are shown in the same plot, allowing a better comparison of how the parameters changed (Figure 7b). Note that when no evidences are selected, only the original marginal distribution, $p(Q = q)$, is displayed on clicking or searching a desired node in the search bar. As both the original and updated distributions are cached in the backend, the estimated time for presenting the marginal distribution of a specific node is highly optimized having a constant complexity, which in real time is equivalent to only a couple of seconds.

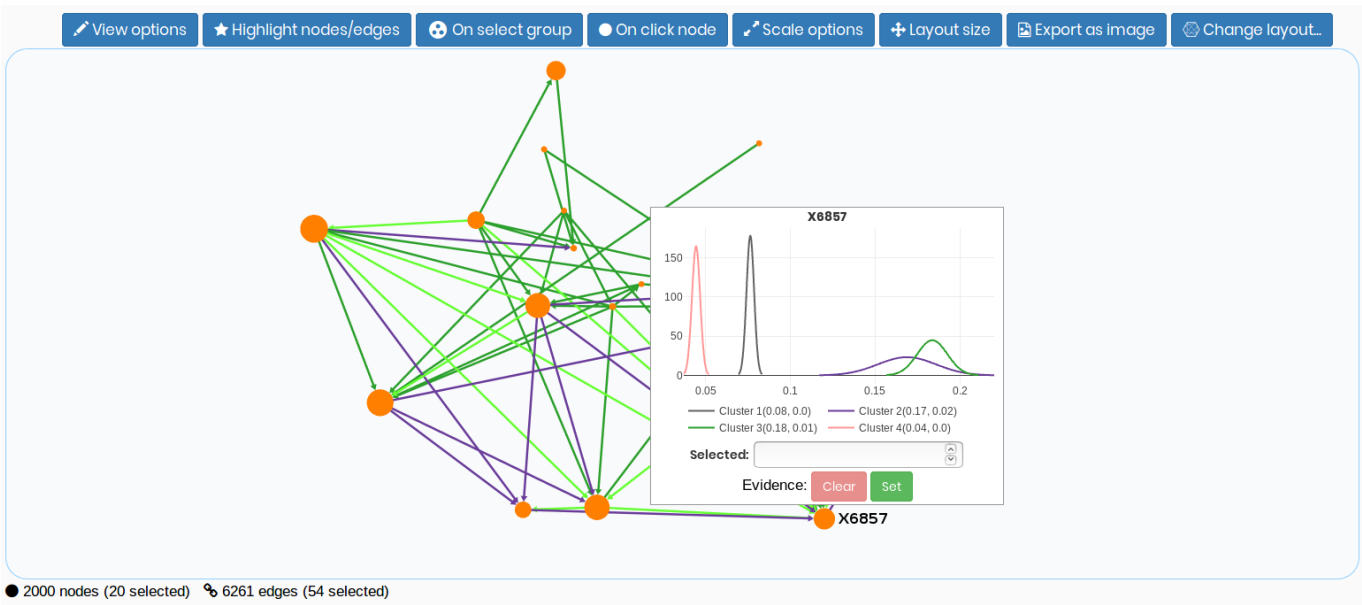
To provide useful insights about the inference effects, we display multiple sorted lists of the query nodes, demonstrating how much their distribution changes according to the KL divergence value, mean variation, and standard deviation variation (Figure 8). When the case groups are created, a list of the multivariate KL divergence values for each group is also be displayed.

In addition, to support another functionality for understanding the graph, we implemented the D-separation criterion following the reachable algorithm described in Koller and Friedman (2009), which can automatically check for conditional independences. Two random variables X and Y are conditionally independent given a random variable Z, if for any assignment of values $X = x$, $Y = y$, $Z = z$, knowing the value of X does not affect the probability of Y when the value of Z is already known, i.e. $P(Y|X, Z) = P(Y|Z)$. Thus, the D-separation algorithm can be particularly useful when we are running inference and want to determine whether some nodes are conditionally independent when some evidence nodes are given.

We have implemented further extensions to support BN-based probabilistic clustering models. The utilized dataset for this use case is also from the Allen Brain Atlas, specifically the one in the Cell Types Database: RNA-Seq Data, which contains single-cell gene expression levels across the human cortex. Therefore, the genes correspond to a set of continuous attributes $\boldsymbol{X} = \{X_1, ..., X_n\}$ for the cell measurements (i.e. the dataset instances).



(a) Network edges for four clusters, each one with a different colour. The upper part of the image also presents the cluster weights. Node sizes are adjusted to highlight the most important nodes with the betweenness centrality algorithm. Nodes colours are according to external metadata to organize them in three groups given their importance.



(b) Selection of a group with the 20 most relevant nodes according to the metadata uploaded file. The plot displays the parameters of gene X6857. Each of the four clusters (different colours), presents a Gaussian distribution. In this example, we can easily notice that the most probable cluster assignation for this gene is cluster 1 (in grey), $p(X6857|C = c_1)$.

Fig. 9: **BN-based probabilistic clustering model of 2000 nodes of the human brain genome.**

In model-based clustering (Fraley and Raftery, 2002), it is assumed that the data follow a joint probability distribution, $P(\boldsymbol{X})$, which can be expressed as a finite mixture of K components. This implies that each mixture component, $P(\boldsymbol{X}|C=c)$, refers to the CPD of $\boldsymbol{X}$ variables given a cluster, c, where the hidden cluster, c, has its own probability distribution, $P(C=c)$. Thus,

$$P(\boldsymbol{X}) = \sum_{c=1}^{K} P(C=c)P(\boldsymbol{X}|C=c). \tag{3}$$

Learning the parameters for this mixture model requires a more advanced technique than MLE, because the cluster variable is hidden. Therefore, we learn the parameters (mixture weights $P(C=c)$ and the CPD parameters, i.e. $P(\boldsymbol{X}|C=c)$) with the expectation maximization algorithm (Dempster et al., 1977) because it can handle incomplete data.

In genomics it is typically assumed that $P(\boldsymbol{X}|C=c)$ follows a multivariate Gaussian distribution, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Hence, the parameters are the mixture weight vector, $\boldsymbol{\pi}$, and the multivariate Gaussian distribution parameters, i.e. the mean vector $\boldsymbol{\mu}$, and the covariance matrix, $\boldsymbol{\Sigma}$.

Numerous genes require a high-dimensional model, which can lead to major computational problems, in terms of both memory and time. For instance, we would have to work with $\boldsymbol{\Sigma}$, which is an $n \times n$ matrix, where n is the number of $\boldsymbol{X}$ variables (genes in this case). To reduce the computational complexity and improve the interpretability, we can factorize this expression to encode the conditional independences between the $\boldsymbol{X}$ variables in a cluster. This allows different graphical models for different clusters, because the relationships between the $\boldsymbol{X}$ variables are conditioned on each cluster as

$$P(\boldsymbol{X}|C=c) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i), C=c) \tag{4}$$

To represent this, we display each graph corresponding to $P(\boldsymbol{X}|C=c)$ in the same BN, colouring the edges with different colours for each cluster (Figure 9a). Selection tools are also implemented to show/hide the different cluster edges and filter them (Figure 9b).

Finally, we express the joint probability distribution of $\boldsymbol{X}$ (Eq. 3) factorized according to Eq. 4. We call this BN-based probabilistic clustering (Pham and Ruz, 2009),

$$P(\boldsymbol{X}) = \sum_{c=1}^{K} P(C=c) \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i), C=c) \tag{5}$$

Therefore, inference can be performed on each graph corresponding to a cluster without affecting the other cluster CPDs. For instance, we can fix the evidence for the distribution of a gene, as $X_i = e$, given a cluster $C=c$, where e is a scalar value, and then query another gene to determine how its CPD for that cluster has changed, $P(X_j|C=c, X_i=e)$.

The obtained BN can be exported as an SVG image or as a CSV file containing the graph information about the arcs between the nodes. This exported file can be loaded subsequently in another session to continue working. Finally, it is important to acknowledge that the user data in a session remains in our servers for 48 h since the last modification of the data. This limit is imposed by our hardware limitations. To overcome this limitation, a user can always create new sessions, and the data will be stored again for 48 h. Users are also encouraged to deploy their own server instance to modify the framework according to their needs.

4 Discussion

We believe that the ease of use will be helpful in initiating collaborations between experts of multiple disciplines. This will be extremely important for the adoption of these models by experts of other disciplines who are not used to programming or software engineering, such as some neuroscientists or physicians.

Here, we review some possible future use cases for

which we believe that this tool can be of great interest. The first example could be the use of a private server instance in closed local networks environments, such as hospitals, clinical laboratories, or companies. A workflow could be easily designed to have a clear pipeline to process the data with machine learning techniques. New data sources connections could be implemented to automatically plug into the data acquisition machines. In addition, some type of specific preprocessing for the data could be implemented in NeuroSuites (e.g. for genomic data it could be the removal of irrelevant genes and the inclusion of domain knowledge about the most important genes). Further, the experts could analyse the data with the NeuroSuites-BNs framework. The web characteristics of the frameworks would make the tool available in a web browser for each employee in the local network without the need of installing the software on their computer.

Finally, we also believe this simplicity could be a great aid for educational purposes when teaching BNs allowing the theorical properties to be shown in a dynamic environment.

The framework aims to be a complete product; however, this is an extremely large research field, and at the time of writing this paper it does not include all the existing state-of-the-art functionalities. Its extensibility properties can make it possible to include numerous extensions and implement new functionalities.

A useful implementation to be included would be some inference algorithms for discrete BNs. We have provided the support to learn and visualize discrete parameters in BNs. However, we have not included yet any inference algorithm for them owing to the development time constraints and the difficulty to visualize the changes in the parameters when there are many parameters per node and numerous nodes. Moreover, massive datasets in various neuroscience fields, such as genomics and electrophysiology, comprise only continuous features.

Another interesting extension would be the inclusion of dynamic BNs (Murphy, 2002). The steps to implement this would be similar to the ones described in the last section to include BN-based clustering models. However, there would be an increased complexity to visualize the network for each timeframe and for performing new types of inferences (e.g. filtering, smoothing, etc.).

Finally, we want to highlight that NeuroSuites also offers different tools, such as morphological reconstructions and microscopy data, for other neuroscience domains. However, although this framework is designed focusing on the neuroscience field, many other tools can also be used in other research fields. Developers can modify the platform to target a different research field. However, it is also important to note that no modifications are needed if the user wants to upload his own dataset and learn a probabilistic graphical model and interpret it, despite the neuroscience background theme of the website. For instance, the use case that we followed here needs a specific BN structure learning algorithm designed for genomics (FGES-Merge) along with all the visualization tools for understanding its massive network. However, for other domains, where datasets are relatively smaller, other algorithms could also be applied.

Data availability

Our production server on https://neurosuites.com can be freely accessed, where all the futures updates will be live. We also provide access to the NeuroSuites source code repository in https://gitlab.com/mmichiels/neurosuite. The BNs used in the examples for showcasing NeuroSuites-BNs can be found in https://gitlab.com/mmichiels/fges_parallel_production/tree/master/BNs_results_paper

Authors' contributions

Mario Michiels designed the software architecture, developed the software and wrote the manuscript. Pedro Larrañaga and Concha Bielza conceived the project, oversaw the development process, contributing with new ideas and corrections, and reviewed the manuscript. All authors gave final approval for publication and agree to be held accountable for the work performed therein.

Competing interests

We declare we have no competing interests.

Funding

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement No. 785907 (HBP SGA2) and from the Spanish Ministry of Economy and Competitiveness through the TIN2016-79684-P project.

Acknowledgments

The authors would like to thank Sergio Paniego for his help in the development of the NeuroSuites-BNs visualization tool, Nikolas Bernaola for his assistance in programming the continuous inference engine for BNs, and Fernando Rodriguez-Sanchez for his research in BN-based probabilistic clustering models and his help in reviewing that section in this paper.

References

Ankan, A. and Panda, A., (2015). pgmpy: Probabilistic Graphical Models using Python. In *Proceedings of the 14th Python in Science Conference*. 6–11. doi: 10.25080/majora-7b98e3ed-001

Aragam, B., Gu, J., and Zhou, Q., (2019). Learning large-scale Bayesian Networks with the sparsebn package. *Journal of Statistical Software* 91, 1–38. doi:10.18637/jss.v091.i11

Ascoli, G. A., Maraver, P., Nanda, S., Polavaram, S., and Armañanzas, R., (2017). Win-win data sharing in neuroscience. *Nature Methods* 14, 112–116. doi: 10.1038/nmeth.4152

Asilomar AI Principles, (2017). Principles developed in conjunction with the 2017 Asilomar conference. In *Benevolent AI 2017*

Benjumeda, M., Bielza, C., and Larrañaga, P., (2019). Learning tractable Bayesian networks in the space of elimination orders. *Artificial Intelligence* 274, 66–90. doi:10.1016/j.artint.2018.11.007

Bernaola, N., Michiels, M., Larrañaga, P., and Bielza, C., (2019). FGES-Merge. Available online at: https://gitlab.com/mmichiels/fges_parallel_production. (Acessed 14th August 2019)

Bielza, C. and Larrañaga, P., (2014). Bayesian networks in neuroscience: A survey. *Frontiers in Computational Neuroscience* 8, 131. doi:10.3389/fncom.2014.00131

Blondel, V. D., Guillaume, J. L., Lambiotte, R., and Lefebvre, E., (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, P10008. doi: 10.1088/1742-5468/2008/10/P10008

Bouchard, K. E., Aimone, J. B., Chun, M., Dean, T., Denker, M., Diesmann, M., et al., (2016). High-Performance Computing in Neuroscience for Data-Driven Discovery, Integration, and Dissemination. *Neuron* 92, 628–631. doi:10.1016/j.neuron.2016.10.035

Cantarelli, M., Marin, B., Quintana, A., Earnshaw, M., Court, R., Gleeson, P., et al., (2018). Geppetto: A reusable modular open platform for exploring neuroscience data and models. *Philosophical Transactions of the Royal Society B: Biological Sciences* 373, 20170380. doi:10.1098/rstb.2017.0380

Celery. Celery: Distributed task queue. Available online at: http://www.celeryproject.org/

Chickering, D. M., Geiger, D., and Heckerman, D., (1994). *Learning Bayesian networks is NP-hard*. Tech. rep., MSR-TR-94-17, Microsoft Research, Advanced Technology Division, Microsoft Corporation, Redmond, WA

Chippada, B. ForceAtlas2 for Python. Available online at: https://github.com/bhargavchippada/forceatlas2. (Acessed 14th August 2019)

Chow, C. K. and Liu, C. N., (1968). Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory* 14, 462–467. doi:10.1109/TIT.1968.1054142

Conrady, S. and Jouffe, L., (2013). Introduction to Bayesian Networks BayesiaLab. *Bayesia SAS, USA*

Cooper, G. F., (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence* 42, 393–405

Csardi, G. and Nepusz, T., (2006). The igraph software package for complex network research. *InterJournal* Complex Sy, 1695

Dagum, P. and Luby, M., (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence* 60, 141–153

Dalcin, L. mpi4py: Python bindings for MPI. Available online at: https://github.com/mpi4py/mpi4py. (Acessed 14th August 2019)

Dempster, A. P., Laird, N. M., and Rubin, D. B., (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1–22

Django Software Foundation, (2013). The Web framework for perfectionists with deadlines — Django. Available online at: https://www.djangoproject.com/

Druzdzel, M. J., (1999). SMILE: Structural Modeling, Inference, and Learning Engine and GeNIe: A development environment for graphical decision-theoretic models. In *AAAI/IAAI*. 902–903

Fraley, C. and Raftery, A. E., (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association* 97, 611–631

Fruchterman, T. M. and Reingold, E. M., (1991). Graph drawing by force-directed placement. *Software: Practice and Experience* 21, 1129–1164. doi: 10.1002/spe.4380211102

Gautier, L. rpy2. Available online at: https://rpy2.bitbucket.io/

Goodman, B. and Flaxman, S., (2017). European Union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine* 38, 50–57

Graphistry. PyGraphistry: A library to extract, transform, and visually explore big graphs. Available online at: https://github.com/graphistry/pygraphistry. (Acessed 14th August 2019)

Gunning, D., (2017). Explainable artificial intelligence (XAI). Available online at: https://www.darpa.mil/program/explainable-artificial-intelligence. (Acessed 10th October 2019)

Hagberg, A. A., Schult, D. A., and Swart, P. J., (2008). *Exploring Network Structure, Dynamics, and Function using NetworkX*. Tech. rep., Los Alamos National Lab (LANL)

Hawrylycz, M. J., Lein, E. S., Guillozet-Bongaarts, A. L., Shen, E. H., Ng, L., Miller, J. A., et al., (2012). An anatomically comprehensive atlas of the adult human brain transcriptome. *Nature* 489, 391

Hayashi, S. and Pestilli, F., (2017). Reproducible neuroimaging via open cloud services: data upcycling to advance discovery in network neuroscience. Available online at: https://brainlife.io/

Højsgaard, S., (2012). Graphical independence networks with the gRain package for R. *Journal of Statistical Software* 46, 1–26

Irrthum, A., Wehenkel, L., Geurts, P., and Others, (2010). Inferring regulatory networks from expression data using tree-based methods. *PloS one* 5, e12776

Jacomy, A. and Plique, G. Sigmajs. Available online at: http://sigmajs.org/. (Acessed 25th November 2019)

Jacomy, M., Venturini, T., Heymann, S., and Bastian, M., (2014). ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PLoS ONE* 9, e98679. doi: 10.1371/journal.pone.0098679

Jones, E., Oliphant, T., and Peterson, P. SciPy: Open source scientific tools for Python. Available online at: http://www.scipy.org/

Kashcha, A. VivaGraphJS: Graph drawing library for JavaScript. Available online at: https://github.com/anvaka/VivaGraphJS. (Acessed 14th August 2019)

Koller, D. and Friedman, N., (2009). *Probabilistic Graphical Models - Principles and Techniques*. MIT press

Koski, T. J. T. and Noble, J., (2012). A review of Bayesian networks and structure learning. *Mathematica Applicanda* 40

Koutsofios, E. and North, S., (1991). *Drawing Graphs with Dot*. Tech. rep., 910904-59113-08TM, ATT Bell Laboratories, Murray Hill, NJ

Lacave, C., Luque, M., and Diez, F. J., (2007). Explanation of Bayesian networks and influence diagrams in Elvira. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37, 952–965

Leitner, F., Bielza, C., Hill, S. L., and Larrañaga, P., (2016). Data publications correlate with citation impact. *Frontiers in Neuroscience* 10, 419. doi:10.3389/fnins.2016.00419

Liu, F., Zhang, S.-W., Guo, W.-F., Wei, Z.-G., and Chen, L., (2016). Inference of gene regulatory network based on local bayesian networks. *PLoS computational biology* 12, e1005024

Luengo-Sanchez, S., Larranaga, P., and Bielza, C., (2019). A Directional-Linear Bayesian Network and Its Application for Clustering and Simulation of Neural Somas. *IEEE Access* 7, 69907–69921. doi:10.1109/ACCESS.2019.2918494

Madsen, A. L., Jensen, F., Kjærulff, U. B., and Lang, M., (2005). The Hugin Tool for probabilistic graphical models. *International Journal on Artificial Intelligence Tools* 14, 507–543. doi:10.1142/S0218213005002235

Madsen, A. L., Jensen, F., Salmerón, A., Langseth, H., and Nielsen, T. D., (2017). A parallel algorithm for Bayesian network structure learning from large data sets. *Knowledge-Based Systems* 117, 46–55

McKinney, W., (2010). Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, eds. S. van der Walt and J. Millman. 51–56

Merkel, D., (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal* 2014, 2. doi:10.1097/01.NND.0000320699.47006.a3

Murphy, K., (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. thesis, University of California

Nair, A., Chetty, M., and Wangikar, P. P., (2015). Improving gene regulatory network inference using network topology information. *Molecular BioSystems* 11, 2449–2463. doi:10.1039/c5mb00122f

Nott, G., (2017). Explainable Artificial Intelligence: Cracking Open the Black Box of AI. *Computer world* 4

Pearl, J., (1988). *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann. doi:10.1016/c2009-0-27609-4

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al., (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, 2825–2830

Pham, D. T. and Ruz, G. A., (2009). Unsupervised training of Bayesian networks for data clustering. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 465, 2927–2948

Piñero, J., Bravo, À., Queralt-Rosinach, N., Gutiérrez-Sacristán, A., Deu-Pons, J., Centeno, E., et al., (2016). DisGeNET: A comprehensive platform integrating information on human disease-associated genes and variants. *Nucleic Acids Research* 45, D833–D839. doi:10.1093/nar/gkw943

Plique, G., (2017). ForceAtlas2 sigmajs plugin. Available online at: https://github.com/jacomyal/

sigma.js/tree/master/plugins/sigma.layout.forceAtlas2. (Acessed 25th November 2019)

PostgreSQL. PostgreSQL: The world's most advanced open source database. Available online at: https://www.postgresql.org/. (Acessed 14th August 2019)

RabbitMQ. RabbitMQ. Available online at: https://www.rabbitmq.com/. (Acessed 25th November 2019)

Rebane, G. and Pearl, J., (1987). The Recovery of Causal Poly-trees from Statistical Data. In *Proceedings of the Third Conference on Uncertainty in Artificial Intelligence*. AUAI Press, UAI'87, 222–228

Robinson, R., (1973). Counting labeled acyclic digraphs. In *New Directions in the Theory of Graphs (Proc. Third Ann Arbor Conf., Univ. Michigan, Ann Arbor, Mich., 1971)*, ed. Academic Press. 239–273

Rudin, C., (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 206–215. doi:10.1038/s42256-019-0048-x

Samek, W., Wiegand, T., and Müller, K.-R., (2017). Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models

Schwarz, G., (1978). Estimating the Dimension of a Model. *The Annals of Statistics* 6, 461–464. doi:10.1214/aos/1176344136

Scutari, M., (2010). Learning Bayesian networks with the bnlearn R Package. *Journal of Statistical Software* 35, 1–22. doi:10.18637/jss.v035.i03

Spirtes, P., Glymour, C. N., Scheines, R., Heckerman, D., Meek, C., Cooper, G., et al., (2000). *Causation, prediction, and search*. MIT press

Sugiyama, K., Tagawa, S., and Toda, M., (1981). Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man and Cybernetics* 11, 109–125. doi:10.1109/TSMC.1981.4308636

Sunkin, S. M., Ng, L., Lau, C., Dolbeare, T., Gilbert, T. L., Thompson, C. L., et al., (2012). Allen Brain Atlas: an integrated spatio-temporal portal for exploring the central nervous system. *Nucleic Acids Research* 41, D996—-D1008. doi:10.1093/nar/gks1042

Sysoev, I., (2004). nginx. Available online at: https://nginx.org/. (Acessed 25th November 2019)

Tenopir, C., Allard, S., Douglass, K., Aydinoglu, A. U., Wu, L., Read, E., et al., (2011). Data sharing by scientists: Practices and perceptions. *PLoS ONE* 6, e21101. doi:10.1371/journal.pone.0021101

Tsamardinos, I., Brown, L. E., and Aliferis, C. F., (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning* 65, 31–78. doi:10.1007/s10994-006-6889-7

Unbit. uWSGI. Available online at: https://uwsgi-docs.readthedocs.io/en/latest/. (Acessed 25th November 2019)

Van Der Walt, S., Colbert, S. C., and Varoquaux, G., (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering* 13, 22–30. doi:10.1109/MCSE.2011.37

Vohra, D., (2016). Apache Parquet. In *Practical Hadoop Ecosystem*, Springer. 325–335. doi:10.1007/978-1-4842-2199-0_8

Walker, D. W. and Dongarra, J. J., (1996). MPI: A standard message passing interface. *Supercomputer* 12, 56–68

Wicherts, J. M., Bakker, M., and Molenaar, D., (2011). Willingness to share research data is related to the strength of the evidence and the quality of reporting of statistical results. *PLoS ONE* 6, e26828. doi:10.1371/journal.pone.0026828

Yuan, C., Lim, H., and Lu, T. C., (2011). Most relevant explanation in bayesian networks. *Journal of Artificial Intelligence Research* 42, 309–352. doi:10.1613/jair.3301