Parallel Graph-Theoretical Analysis Toolkit Manuscript

PAGANI Toolkit aims at accelerating the frequently used but time-consuming algorithms in neuroscience research. We have accelerated the process of brain network analysis (BNA) with NVIDIA GPUs (Graphic processing unit) and multi-core CPUs. The toolbox provides functions for the construction and the analysis of large networks. Network construction is intended for fMRI data using Pearson's correlation. Network analysis is general purposed and includes the calculation of both global metrics (i.e., global clustering coefficient, characteristic path length, modularity, and small-world property) and nodal metrics (i.e., nodal degree, efficiency, betweenness, eigenvector centrality, and participation coefficient). We accelerate Pearson's correlation calculation, APSP and betweenness, eigenvector-based modular detection with GPUs. Other functions are implemented on multi-core CPUs. If you found the platform useful, please cite our paper published on plos one

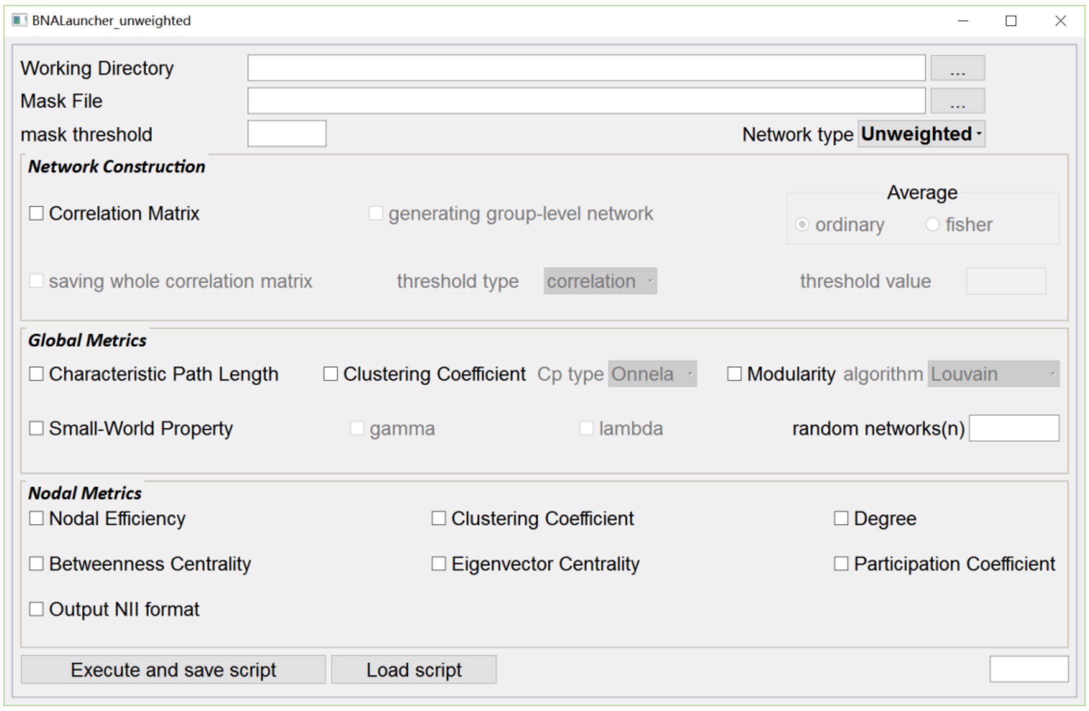
Wang Y. et al, (2013) A Hybrid CPU-GPU Accelerated Framework for Fast Mapping of High-Resolution Human Brain Connectome. PloS one 8:e62789.

Runtime Environment

The new release version of PAGANI Toolkit requires a 64-bit Windows operating system, a PC or server with an NVIDIA GPU supporting CUDA. The computation part was packaged into independent executable files for different functions while the GUI was meticulously designed to flexibly generate scripts for performing the computations in batch.

The old version is called parabna containing the source code files for both 64-bit Windows, and Linux, requiring additionally the latest CUDA Toolkit (http://www.nvidia.com/content/cuda/cuda-downloads.html). We have tested the win64 version on NVIDIA GTX 580 GPU or NVDIA GeForce GTX 850M with CUDA Toolkit v7.0 and Windows 7，Windows 8 and Windows 10 operating system, and Linux version on NVIDIA Titan GPU with CUDA Toolkit 7.0 and CentOS 6.5 operating system. To get started with CUDA, please follow NVIDIA CUDA Getting Started Guide for Microsoft Windows
(http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/index.html)
or NVIDIA CUDA Getting Started Guide for Linux
(http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html).

Graphic User Interface



Network type: Unweighted
threshold type: correlation
Cp type: Onnela
algorithm: Louvain

Fig1. GUI of PAGANI platform

Review

We have already provided a graphic user interface(GUI) now with which researchers can quickly get started and have a better user experience. However, the instruction of command line operation still remains for compatibility with different preference. For details, please see 'Function Interface' section.

First of all, users should specify the working directory and mask file at the top of GUI. All input imaging files (resting fMRI) should be placed directly under the working directory in NIfTI format, and all results will be saved in the same directory using similar names with the data file with only a few modifications. The mask file should also be in NIfTI format, the data type can be float or unsigned char (more details on nifti website, http://nifti.nimh.nih.gov/nifti-1/documentation/nifti1fields/nifti1fields_pages/datatype.html). The mask file and corresponding threshold select voxels of interest, for example, voxels belonging to the gray matter in functional network analysis. If data type of mask file is Binary, this value can be arbitrary from 0~1. Either binary or weighted network type can be supported in PAGANI toolkit. Some algorithms currently are not enable in weighted network such as betweenness centrality, eigenvector-based modular detection, etc.

The workflow of PAGANI Toolkit can be summarized into two main steps: network construction and network analysis. Thereinto, the metrics of network analysis include two sorts: global, nodal.

Network Construction

Briefly, the input for the network construction should be preprocessed fMRI data. GPU-based acceleration algorithm is applied to calculate Pearson's correlations between every pair of voxels within an input mask, resulting the voxel-level functional networks. Users can select either binary or weighted network type for further analysis according to whether they take connectivity strength into account.

The emerged tips will give detailed implications for corresponding option, when users put the mouse arrow on certain bar.

Metrics

Two sorts of metrics are provided: global metrics and nodal metrics. For details see table below:

Network metric	**Descriptions**	**Definitions**
Global network metrics		
Clustering coefficient (*Cp*)	A measure of local clustering and closeness of a network	Seen next table
Characteristic path length (*Lp*)	A measure of integration and global routing efficiency of a network	$Lp = \frac{n(n-1)}{2\sum_{i \neq j \in N} d_{ij}^{-1}}$ where d_{ij} is the shortest path length between *i* and *j*.
Gamma (*γ*)	The ratio of *Cp* of a network to those of random networks with identical degree distribution	$\gamma = Cp / Cp_{rand}$
Lambda (*λ*)	The ratio of *Lp* of a network to those of random networks with identical degree distribution	$\lambda = Lp / Lp_{rand}$
Sigma (*σ*)	The extent of small-world property of a network, indicating high local closeness and global communicational integration	$\sigma = \gamma / \lambda$
Modularity (*Q*)	A measure of the extent to which a network can be partitioned into tightly intra-connected and sparsely inter-connected modules.	$Q = \frac{1}{2m} \sum_{i \neq j \in N} (a_{ij} - \frac{k_i k_j}{2m}) \delta(g_i, g_j)$ where *m* is the number of network edges, g_i is the module to which node *i* belongs to, and $\delta(g_i, g_j)$ is 1 if $g_i = g_j$, and 0 otherwise.
Nodal (voxel-wise) metrics		
Degree (k_i)	The number of links connected directly to a node	$k_i = \sum_{j \in N} a_{ij}$
Efficiency (e_i)	A measure of efficiency for a node communicating with the other nodes	$e_i = \frac{1}{n-1} \sum_{j \in N, j \neq i} d_{ij}^{-1}$

Betweenness (b_i)	A centrality measure of a node in the communications between other nodes	$b_i = \frac{2}{(n-1)(n-2)} \sum_{j \neq i \neq h \in N,} \frac{\theta_{jh}(i)}{\theta_{jh}}$ where θ_{jh} is the number of shortest paths between j and h, $\theta_{jh}(i)$ is the number of shortest paths between j and h that pass through i.
Eigenvector Centrality(g_i)	Eigenvector centrality is a measure of the influence of a node in a network.	$g_i = \frac{1}{\lambda_{max}} \sum_{j \in N} a_{ij} g_j$ Where λ_{max} is the most positive eigenvalue of adjacent matrix of network.
Participation coefficient (p_i)	A measure of the ability of a node in keeping the communication between different modules.	$p_i = 1 - \sum_{s=1}^{N_M} \left(\frac{k_{is}}{k_i} \right)^2$ Where N_M is the number of modules, k_{is} is the degree of node i to nodes in module s.

Note: N is the set of all nodes in the network, and n is the number of nodes. a_{ij} is the element of adjacency matrices indicating the connection condition between node i and j, a_{ij}=1 if node i is directly connected to node j, otherwise a_{ij}=0.

At the bottom of GUI, ‘Execute and Save Script’ button saves all settings in a batch script and execute the batch file. ‘Load script’ will load previous settings saved in an existing script.

Function Interface

1. CUCorMat()

Note that this new version of CUCorMat() has made a few changes of parameters.

```
CUCorMat Dir_for_BOLD Mask_Path threshold_for_mask(0~1) to_average(yf/y
n/bf/bn/n) to_save_cormatrix(y/n) threshold_type(r/s) threshold_for_corr
eletaion_coefficient(s)(0~1)
```

This function constructs correlation matrices from BOLD signals on GPU. There are at least six arguments: 1) a directory containing NII files of BOLD signals; 2) the path of corresponding mask file (data type: float or binary, can be recognized automatically); 3) A threshold to select valid voxels according to the mask; 4) a flag indicating whether or not the correlation matrices are to be averaged; 5) a flag indicating whether or not the original results are to be saved; 6) threshold_type indicates that the binary threshold is for correlation values or sparsities; 7) a list of threshold value, at least one threshold for binarizing the correlation coefficients. The outputs are connection matrices in .csr files and upper-triangle matrices in .cormat files, if enabled.

Both weighted and unweighted networks are supported for function CUCorMat. Note that the output directory is different with different type of network. For detail information see the following table and example.

Parameter	Meaning
Dir_for_BOLD	The input directory containing NII files of BOLD signals. In the directory, there has to be a file named ‘mask.nii’, which gives the probability of each voxel belonging to the gray matter. All the NII files must be **little-endian (ieee-le)** and the data type must be **32-bit real** (single-precision floating point numbers).
MASK_PATH	The path of mask file.
threshold_for_mask	A threshold to select valid voxels from ‘mask.nii’. Data type: float or boolean, which can be recognized automatically. If data type of mask.nii is Boolean or unsigned char, this value can be arbitrary from 0~1.
to_average	A flag indicating whether or not the correlation matrices are to be averaged. If average_flag = ‘yf’ or ‘yn’, all correlation matrices are averaged and only one adjacency matrix is generated; If average_flag = ‘n’, each NII file for BOLD signals corresponds to an adjacency matrix; If average_flag = ‘bf’ or ‘bn’, both the individual adjacency matrices and the average adjacency matrix are generated. The second character ‘f’ or ‘n’ indicates whether fisher transformation is used in the process of averaging.
to_save_cormatrix	A flag indicating whether or not the original correlation matrices are to be saved. If to_save_cormatrix=’y’, both the original correlation matrices and the binarized csr results will be saved under input directory (Dir_for_BOLD). If to_save_cormatrix=’n’, only binarized csr results will be saved. These matrices are stored in *.cormat* files.
threshold_type	A parameter indicating all correlation matrices are thresholded by the same correlation value or the same sparsity. For correlation threshold, threshold_type = “r”; For sparsity threshold, threshold_type = “s”.
threshold_for_ correletaion_coefficient	A set of thresholds for binarizing the correlation coefficients. Each threshold will generate a set of outputs.

Example :
The following command

```
./CUCorMat ../../data/ ../../mask.nii 0.2 yn n r 0.25 0.3 0.35
```

generates the output files in the directory ../../data/unweighted for unweighted network and the directory ../../data/weighted for weighted network.

2. CUBFW_Lp(), CUBFS_Lp(), BFS_MulCPU()

```
CUBFS_Lp input_dir num_of_random_networks
#This function is not recommended as it is not stable.
BFS_MulCPU input_dir num_of_random_networks output_type
CUBFW_Lp input_dir num_of_random_networks output_type
```

These three functions calculate the characteristic path length (Lp) of the network and compare with K (user specified) random networks on GPU and multi-core CPU. The input arguments are 1) a directory containing .csr files; 2) the number of random networks for comparison; 3) the output metrics of the function. The corresponding parameter sets are: g, n, l, gn, gl, nl, gnl. Thereinto, 'g' represents the global metric 'characteristic path length', 'n' represents the nodal metric 'nodal efficiency', 'l' represents the global metric 'lambda'. 'gn', 'gl', 'nl', 'gnl' are used on behalf of the combination of these three basic output method above. For instance, 'gnl' indicates that all the three metrics will be saved. The functions will calculate Lp for each network in the input directory. The output is a text file for each input network, storing the Lp results of the brain network and K random networks (one file for each .csr network).

Lp is the harmonic average of All-Pairs-Shortest-Path (APSP), and also the reciprocal of global efficiency. The two functions use different algorithms to calculate APSP. CUBFS_Lp() and BFS_MulCPU() implement the Breadth First Search (BFS) algorithm, which performs well with sparse networks but poorly with dense ones, on GPU and multi-core CPU respectively. The performance of these two functions is comparable with each other. We suggest using the latter one if you have a more than 8 cores CPU. CUBFW_Lp() uses the blocked Floyd-Warshall algorithm (BFW), which outperforms BFS on dense networks. The transition point of the performance of the two algorithms is approximately where network density equals 3%. It means CUBFS_Lp() is recommended if the network density is lower than 3% and CUBFW_Lp() is recommended if otherwise.

Both weighted and unweighted networks are supported for function CUBFW_Lp.

Only unweighted networks are supported for BFS-based functions.

Parameter	Meaning
input_dir	The input directory containing *.csr* binary networks.
num_of_random_networks	The number of random networks with the same degree distribution for comparison. If num_of_random_networks == 0, no comparison is initiated.
output_type	The output type of Lp. The corresponding parameter sets are: g, n, l, gn, gl, nl, gnl. For detail information see text.

Example:

The commands

./CUBFW_Lp ../../data/ 15 gnl (Characteristic path length, nodal efficiency and lambda all are printed)

./BFS_MulCPU ../../data/ 15   nl.   (only output nodal efficiency and lambda)

Note that PAGANI toolkit can automatically select more efficient algorithm among CUBFW_Lp and BFS_MulCPU according to sparsity of network if you use GUI.

3. Cp()

```
Unweighted network:
Cp input_dir num_of_random_networks output_type
Weighted network:
Cp input_dir num_of_random_networks parameter_type output_type
```

This function calculates clustering coefficient (Cp) of the network and compare the results with K (user specified) random networks on CPU. The command format is different for different network types. This is because in weighted network clustering coefficient has two different definitions, which is determined by ‘parameter_type’ option. For weighted network as an example, the input arguments are 1) a directory containing .csr files, 2) the number of random networks for comparison, 3) parameter type: the Cp type in weighted network, and 4) output_type: The output of the function. The corresponding parameter sets is: g, n, k, gn, gk, nk, gnk. Thereinto, ‘g’ represents the average Cp involved in global metrics, ‘n’ represents the Cp involved in nodal metrics, and ‘k’ represents the global metric ‘gamma’. ‘gn’, ‘gk’, ‘nk’, ‘gnk’ are used on behalf of the corresponding combination of these three metrics above to be outputted. For instance, ‘gnk’ indicates all the three metrics will be saved. The function will calculate Cp for each network in the input directory.

Both weighted and unweighted networks are supported for function Cp.

Parameter	Meaning
input_dir	The input directory containing .csr binary networks.
num_of_random_networks	The number of random networks with same degree distribution for comparison. If num_of_random_networks == 0, no comparison is initiated.
parameter_type **(Only for weighted network)**	the Cp type in weighted network. parameter_type=1, Barrat; parameter_type=2, Onnela;
output_type	The output type of parameter. The corresponding parameter sets is: g, n, k, gn, gk, nk, gnk. For detail information see text.

Example:

./Cp ../../data/ 15 gnk   (Unweighted networks. Gamma, global Cp, and nodal Cp are printed)

./Cp ../../data/ 15 2 g   (Weighted networks. Onnela Cp. Only global metric is printed)

Note that two different definitions of clustering coefficient in weighted network can be learned in references below:

`Onnela -- Onnela et al., 2005, Intensity and coherence of motifs in weighted complex networks.`

`Barrat -- Barrat et al., 2009, The architecture of complex weighted networks.`

4. Degree()

```
Degree input_dir
```

This function calculates the degree centrality of the network on CPU. The input is a directory containing .csr files. The functions will calculate degree centrality for each network in the input directory, generating .nm files for each input network. There is only the CPU version.

Both weighted and unweighted networks are supported for function Degree.

Parameter	Meaning
input_dir	The input directory containing .csr binary networks.

Example:

```
./Degree ../../data/
```

5. CUBC()

```
CUBC input_dir
```

This function calculates the betweenness centrality of the network on GPU. The input is a directory containing .csr files. The functions will calculate betweenness centrality for each network in the input directory, generating .nm files for each input network.

Only unweighted networks are supported for CUBC functions.

Parameter	Meaning
input_dir	The input directory containing .csr binary networks.

Example:

```
./CUBC ../../data/
```

6. CUEC()

```
CUEC input_dir
```

This function calculates the eigenvector centrality of the network on GPU. The input is a directory containing .csr files. The functions will calculate eigenvector centrality for each network in the input directory, generating .nm files for each input network.

Both unweighted and weighted networks are supported for CUEC functions.

Parameter	Meaning

input_dir	The input directory containing .csr binary networks.

Example:

```
./CUEC ../../data/
```

7. Louvain_Modularity ()

```
Louvain_Modularity dir_for_csr num_of_random_networks
```

This function calculates Louvain Modularity for each of the *.csr* files in the given directory and compares each of them with several optional random networks. The outputs are *.modu* files in the date directory.

Both weighted and unweighted networks are supported for function Louvain_Modularity.

Parameter	Meaning
input_dir	The input directory containing .csr binary networks.
num_of_random_networks	The number of random networks with same degree distribution for comparison. If num_of_random_networks == 0, no comparison is initiated.

Example:

```
./Louvain_Modularity ../../data/ 15
```

8. PC_CPU()

```
PC_CPU dir_for_csr type_for_participant_coefficient
```

This function calculates participant coefficient for each subject ("*.csr*" files) in the given directory. Each "*.csr*" files should be accompanied with a "*.modu*" files, which has a same file name, in the data directory. The type of participant coefficient is original as default and is normalized by (Nm-1)/Nm where Nm is the number of modules if this option is set as 'n'. The outputs are *.pc* files in the date directory.

Both weighted and unweighted networks are supported for function PC_CPU.

Example:

```
./PC_CPU.exe ../../data/      #(output original participant coefficient)

./PC_CPU.exe ../../data/ n #(output normalized participant coefficient)
```

9. ConvertNII()

```
ConvertNII  input_dir  mask_file  mask_threshold
```

This function convert the nodal metric results saved in .nm files to the standard NII format under the input directory. The inputs are: the input directory, mask file in NII format and a threshold to select the voxels. The mask file and the mask threshold should be the same as those used in network construction with CUCorMat().

Both weighted and unweighted networks are supported for function ConvertNII.

Parameter	Meaning
input_dir	The input directory containing files to be converted to NII format.
mask.nii	The name of NII file giving the probability of each voxel belonging to the gray matter.
mask_threshold	A threshold to select valid voxels from 'mask.nii', if the datatype of mask_nifti file is float. If the datatype is binary, an arbitrary value from 0~1 is available.

Example:

```
./ConvertNII ../../data ../maskdir/mask.nii 0.2
```

Nodal metric files should exist in the specified directory. The output NIfTI files are generated in directory ../../data/ .

File Format

The input files in the first procedure CUCorMat and the output files of our platform are all standard NII files with some extra restrictions. Here we also introduce the format of other related files of our platform for the convenience of some users who may want to process these intermediate results.

1. The original input

4D NIfTI files for BOLD signals, Little-endian (ieee-le) single-precision floating point numbers (float) are required. Mask data stored in NIfTI file, data type can be both binary (unsigned char) or float.

2. CSR

.csr file (Compressed Sparse Row). The first 32-bit integer indicates the length of array *R* , which is the number of voxels *N*+1, followed by *N*+1 32-bit integers of the array *R*. Next is a 32-bit integer indicating the length of array *C*, which equals the number of edges *E*, followed by *E* 32-bit integers of the array *C*. That's all for the unweighted network. If the network is weighted, the final part is a 32-bit integer indicating the length of array V, which also equals the number of edges *E,* followed by *E* 32-bit floats of the array V, indicating the weight of each edge remained after thresholding).

3. Characteristics results

Nodal metrics are firstly stored in .nm files with the first 32-bit integer indicating the number of voxels *N*, followed by *N* float numbers indicating the nodal metric at each voxel. For instance, A file whose name end with _deg.nm represents the corresponding degree of that voxel.

4. Correlation matrices

.cormat files begin with a 32-bit integer indicating the number of following numbers, which equals N(N-1)/2 if N is the number of rows or columns. These correlation coefficients are stored in 32-bit float numbers. Note that correlation matrices are symmetric and we only store the *upper*-triangle part in row-major order.

5. Output NII file

All the nodal metric results in .nm format are converted to standard NII file using ConvertNII().