Parallel Brain Network Analysis Platform Manual

This platform 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 clustering coefficient, characteristic path length, network efficiency, and betweenness centrality (and comparisons to Maslov random networks). We accelerate Pearson's correlation calculation, APSP and betweenness with GPUs. Other functions are implemented on multi-core CPUs. If you found the platform useful, please cite our paper published on 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 win64 version requires a 64-bit Windows operating system, an NVIDIA GPU and the latest CUDA Toolkit (http://www.nvidia.com/content/cuda/cuda-downloads.html). We have tested all the functions on NVIDIA GTX 580 GPU with CUDA Toolkit v5.5 and Windows 7 operating system.

The Linux version requires a Linux operating system, an NVIDIA GPU and the latest CUDA Toolkit (http://www.nvidia.com/content/cuda/cuda-downloads.html). We have tested all the functions on NVIDIA Titan GPU with CUDA Toolkit 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).

Function Interface

1. CUCorMat()

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

```
CUCorMat Dir_for_BOLD threshold_for_mask(0~1) to_average(yf/yn/bf/bn/n)
    to_save_cormatrix(y/n) threshold_type(r/s) threshold_for_correletaio
n_coefficient(s)(0~1)
```

This function constructs correlation matrices from BOLD signals on GPU. There are at least six inputs: 1) a directory containing NII files of BOLD signals and a gray matter

mask file named “mask.nii” (data type: float or boolean, can be recognized automatically), 2) a threshold to select valid voxels, 3) a flag indicating whether or not the correlation matrices are to be averaged, and 4) a flag indicating whether or not the original results are to be saved, 5)threshold_type indicates the binary_threshold is for correlation values or sparsities, 6) at least one threshold for binarizing the correlation coefficients. The outputs are adjacency matrices in .csr files and upper-triangle matrices in .cormat files, if enabled.

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).
threshold_for_mask	A threshold to select valid voxels from ‘mask.nii’. Data type: float or boolean, which 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/ 0.2 yn n r 0.25 0.3 0.35
```

generates the output files in the directory ../../data/ .

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

```
CUBFW_Lp input_dir num_of_random_networks
CUBFS_Lp input_dir num_of_random_networks
BFS_MulCPU input_dir num_of_random_networks
```

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 is 1) a directory containing .csr files, and 2) the number of random networks for comparison. 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 algorithm 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.

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.

Example:

The commands

```
./CUBFW_Lp ../../data/ 15
./CUBFS_Lp ../../data/ 15
```

generates both *.eff* files storing the nodal efficiency and *_Lp.txt* for LP.

3. Cp()

```
Cp input_dir num_of_random_networks
```

This function calculates clustering coefficient (Cp) of the network and compare the results with K (user specified) random networks on CPU. The input is 1) a directory containing .csr files, and 2) the number of random networks for comparison. The functions will calculate Cp for each network in the input directory, generating .cp files for each input network.

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:

```
./Cp ../../data/ 15
```

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 .deg files for each input network. There is only the CPU version.

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 .bc files for each input network.

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

Example:

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

6. ConvertNII()

```
ConvertNII  input_file  mask_file  mask_threshold
```

This function puts the .cp .eff .deg .bc results back to the 3-D matrix and converts these filesto the standard NII format. The inputs are: one of the mentioned files, a file for mask data 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().

Parameter	Meaning
input_dir	The input directory containing files to be converted to NII format.

mask.nii	The NII file path 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. An arbitrary value from 0~1 is available if the datatype is unsigned char.

Example:

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

X.cp should exist in the specified directory. The output file X.cp.nii is generated in directory ../datadir/ .

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.

Example:

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

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 restriction. 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

NII files for BOLD signals and mask data. Little-endian (ieee-le) NII files with single-precision floating point numbers (float) are required.

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*.

3. Characteristics results

.deg .cp .eff .bc .modu .pc, files with the first 32-bit integer indicating the number of voxels *N*, followed by *N* integers (nodal degree, i.e. .deg file, module i.e. .modu file) or float numbers (clustering coefficient (.cp), nodal efficiency (.eff), betweenness centrality (.bc), or participant coefficient (.pc)) representing the corresponding characteristics 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 indexes are 32-bit float numbers. Note that correlation matrices are symmetric and we only stores the *upper*-triangle part.

5. Output NII file

All the result files *.deg .cp .eff .bc* are converted to standard NII file using ConvertNII().