hClustering Commands User Guide

• Introduction

The *hClustering* repository groups a set of algorithms created to generate fully hierarchical characterization of human whole-brain high-res dMRI-based anatomical connectivity data (although with minor source code modifications it could be used with functional or any other kind of data that can be represented as a "fingerprint" vector).

These algorithms are delivered in the form of easy-to-use Linux command-line tools with comprehensive command help information. The source code is fully documented using *Doxygen*, and several additional manuals are included such as a quick-start guide and descriptions of the custom file-formats used.

The original code was developed in the framework of the Whole-Brain Hierarchical clustering project carried out in the Max Planck Institute for Human Cognitive and Brain Sciences in Leipzig, as part of the PhD Thesis: "Whole-brain cortical parcellation: A hierarchical method based on dMRI tractography" by David Moreno-Dominguez. For this release, the code has been documented, upgraded and streamlined to be more easy to use and accept both the neuroimaging de-facto standard Nifti (.nii) and *Lipsia* neuroimaging processing package Vista (.v) formats.

For further information on the underlying algorithms and research done with this code refer to:

- Moreno-Dominguez, D., Anwander, A., & Knösche, T. R. (2014).
  A hierarchical method for whole-brain connectivity-based parcellation.
  Human Brain Mapping, 35(10), 5000-5025. doi: http://dx.doi.org/10.1002/hbm.22528

- Moreno-Dominguez, D. (2014).
  Whole-brain cortical parcellation: A hierarchical method based on dMRI tractography.
  PhD Thesis, Max Planck Institute for Human Cognitive and Brain Sciences, Leipzig.
  ISBN 978-3-941504-45-5.

Whole-Brain Connectivity-Based Hierarchical Parcellation Project
David Moreno-Dominguez
d.mor.dom@gmail.com
moreno@cbs.mpg.de
www.cbs.mpg.de/~moreno

- **Index**

Prior requirements

Install if necessary the following library packages:

Boost [libboost-all]:	Tested with version 1.48, higher versions should also work.
via [libviaio]:	Provided as part of the repository. For vista files handling.
nifti-1 [libniftiio]:	For nifti files handling.
lznz [libznz]:	Required by nifti library.

Clone and compile the *hClustering* code from GitHub:
https://github.com/dmordom/hClustering.git.

The minimum data required to use the *hClustering* commands is listed below:

- A 3D white matter mask (the one used as tracking target space).
- A set of probabilistic tractograms in 1D compact form (as many elements as voxels in the white matter mask). *vdconnect* tractography directly outputs tractograms in compact form (in vista format). If available tracts are 3D nifti images they can be transformed to compact form with the full2compact command from this repository.
- An ASCII roi file with the seed coordinates, image size and streamline number information (see file format guide for details).

These (and more) data can be automatically generated by the dMRI preprocessing pipeline available at https://github.com/dmordom/dmri_prepro_nipype.git. For more information refer to the pipeline repository readme and the quick-start guide.

Command workflow

This section will briefly explain the general workflow to follow when using the *hClustering* commands.

The first and main step is to build the hierarchical trees. The centroid-neighbourhood tree (the centre around which our methods are tailored) can be computed directly from the mask+roi+tracts data using the ***buildctree*** command. Alternatively graph-linkage trees can be computed using the ***buildgraphtree*** command over the distance pairwise tractogram-distance matrix (in turn, this matrix is obtained with the ***distmatrix*** command).

If tree quality (in the sense of how much information from the distance matrix is kept in the tree) wishes to be studied, we need only use the ***cpcc*** command (this command requires both the tree output file and the pairwise distance matrix). In order to obtain a baseline level for the *CPCC* quality values in the absence of structure, a set of artificial tractograms yielding a uniformly random matrix can be created with the ***randtracts*** command.

From these, the ***buildrandctree*** method (and a *cpcc* over the result) should be used to obtain a centroid-tree baseline (in the case of a baseline for graph trees, we need only generate a distance matrix from these tracts and graph trees from this matrix using the normal commands).

For further tree processing (mainly linear node-collapse and debinarization) ***processtree*** command should be used. And with ***partitiontree***, automatic best-quality partitions at all granularity levels can be obtained for each tree.

The next and final step regarding individual trees is tree visualization and interactive exploration. This can be done through the "***Hierarchical Clustering***" module developed and integrated in the *OpenWalnut* framework. This program can be downloaded from www.openwalnut.org. For more information refer to the *OpenWalnut* module user guide and the *OpenWalnut* website. This module requires as only necessary input the tree file, but a matching T_1 or FA image and *Freesurfer* surface are also recommended for best visualization.

If tree comparison is to be performed the first step would be to calculate the mean tractograms for the meta-leaves of each tree (the maximum effective granularity level). For this is required that the trees are computed through the centroid-nbhood method (the graph methods implemented here do not produce meta-leaves), and the meta-leaf (base-node) tract to be generated using the ***treetracts*** command.
Once this is done the tractograms should be blown to full 3D images with the ***compact2full*** command, and then warped to a common space with the registration program of choice (like *ANTS* or *FNIRT*). After this, the ***comparetrees*** command can be used. (**NOTE**: in case of comparing trees built over the same dataset and not from different subjects, meta-leaf tractograms are not necessary and the *comparetrees* command can be used directly). This will perform the tree-matching and calculate the tree similarity values.
Right hemisphere trees can be compared to their left-hemisphere counterparts with the help of the ***fliptracts*** and ***fliptree*** commands to previously flip the tractograms and tree from the right hemisphere files respectively to overlap in the space of the left hemisphere, and then execute *comparetrees* over the new data.

If two trees that have been previously matched are then partitioned independently, the partition labels can be matched using the ***matchpartition*** command (also, a best matching partition in a matched tree can be generated from a partition in the other tree).

Finally other auxiliary commands are available to provide further support. These are ***tractdist***, ***pairdist***, ***full2compact***, ***compact2full***, ***cmpct2vista***, ***vista2cmpct***, ***image2tree*** and ***surfprojection***. More information on them can be obtained from the command help.

In the next section, the information of the command help from each algorithm is provided for completeness (in the same order as they were mentioned in this section).

- **buildctree**

Build a centroid hierarchical tree from a set of seed voxels and their corresponding tractograms.

*** Arguments:**

--version:	Program version.
-h --help :	Produce extended program help message.
-r --roi-file :	A text file with the seed voxel coordinates and the corresponding tractogram index (if tractogram naming is based on Index rather than coordinates).
-I --inputf :	Input data folder (containing the compact tractograms).
-T --tempf :	Mean tractogram folder location for centroid method. Will be used as temporary storage for intermediate tractogram results.
-O --outputf :	Output folder where tree files will be written.
[-t --threshold]:	Number of streamlines relative to the total generated that must pass through a tract voxel to be considered for tract similarity. (i.e.: minimum value of a normalized probabilistic tract in natural units to be considered above noise). Valid values: [0,1) Use a value of 0 (default) if no thresholding is desired.
[-d --maxnbdist]:	Maximum dissimilarity a seed voxel tract must have to its most similar neighbour not be discarded. Valid values: (0,1] Use a value of 1 (default) if no discarding is desired.
[-c --cnbhood]:	Use centroid method with C neighbourhood level. Valid values: 6, 18, 24(default), 32, 96, 124.
[-S --basesize]:	Merge homogeneous base nodes of size S. (mutually exclusive with -N option). Default: 0 (no homogeneous merging).
[-N --basenum]:	Grow N homogeneous base nodes. (mutually exclusive with -S option). Default: 0 (no homogeneous merging).
[--nolog]:	Use linear normalization. Use if input tracts are only linearly normalized instead of logarithmically normalized.
[-v --verbose]:	Verbose output (recommended).
[--vista]:	Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[-m --cache-mem]:	Maximum amount of RAM memory (in GBytes) to use for temporal tractogram cache storing. Valid values [0.1,50]. Default: 0.5.
[-k --keep-disc]:	Keep discarded voxel information in a specialised section of the tree.
[--debugout]:	Write additional detailed outputs meant to be used for debugging.
[-p --pthreads]:	Number of processing threads to run the program in parallel. Default: use all available processors.

*** Usage example:**

```
buildctree -r roi_lh.txt -s 5000 -I tractograms/ -T /tmp/tracts -O results/ -t 0.001 -d 0.1 -c 26 -N 1000 -k -m 2 -v
```

*** Outputs (in output folder defined at option -O):**

- *'cXX.txt'* :	(where XX is the neighbourhood level defined at option -c) Contains the output hierarchical tree.
- *'baselist.txt'*:	Contains the IDs of the nodes corresponding to meta-leaves (and forming the maximum granularity level).
- *'success.txt'* :	An empty file created when the program has successfully exited after completion (to help for automatic re-running scripting after failure).
- *'buildctree_log.txt'*:	A text log file containing the parameter details and in-run and completion information of the program.

[extra outputs when using --debugout option]

- *'cXX_bin.txt'*:	Binary-branching hierarchical tree before tree processing.
- *'cXX_bin_nmt.txt'* :	Non-monotonic tree after monotonicity correction.
- *'baselist_bin.txt'*:	Meta-leaves (base nodes) list with IDs corresponding to the binar tree files.
- *'cXX_bin_nmt_granlimit.txt':*	Non-monotonic hierarchical after granularity limitation (and meta-leaf formation).
- *'cXX_Down.txt'*:	Processed hierarchical tree but using a lower-limit monotonicity correction algorithm rather than the standard weighted one.
- *'cXX_Up.txt'*:	Processed hierarchical tree but using a Higher-limit monotonicity correction algorithm rather than the standard weighted one.
- *'cXX_[*]debug.txt'*:	Versions of the counterpart files without suffix with redundant information for debugging purposes.
- *'baselist_bin_10k.txt'*:	IDs of the peak nodes at the building point where only 10k active node remained.
- *'compact_[ID/COORD].nii(.v)'*:	Compact tractogram corresponding to the tree root node (mean tractogram of all seeds included in the final tree).

- ## distmatrix

Compute a pairwise distance matrix between seed voxel compact tracts. Matrix will be divided in sub-blocks for easier & safer computing and storing.

* Notes:

- As matrix will be symmetrical only upper triangle is computed.
- Distance metric used is normalized dot product.
- Memory and CPU heavy.

* Arguments:

```
     --version     :   Program version.
  -h --help        :   Produce extended program help message.
  -r --roi         :   File with the seed voxel coordinates and corresponding tractogram IDs.
  -I --inputf      :   Input data folder (containing the seed voxel compact tractograms).
  -O --outputf     :   Output folder where distance matrix block files will be written.
[ -t --threshold ]:    Number of streamlines relative to the total generated that must pass through a tract voxel to be considered
                           for tract similarity. (i.e.: minimum value of a normalized probabilistic tract in natural units to be
                           considered above noise). Valid values: [0,1) Use a value of 0 (default) if no thresholding is desired.
[ -b --blocksize ]:    Desired size (in number of elements per row/column) of the blocks the distance matrix will be subdivided in.
                           Choose 0 for maximum size according to available memory. Default: 5000.
[    --start     ]:    A pair of row-column integers indicating the first block where to start the process.
                           Previous blocks will not be computed.
[    --finish    ]:    A pair of row-column integers indicating the last block where to finish the process.
                           Posterior blocks will not be computed.
[ -v --verbose   ]:    Verbose output (recommended).
[ -V --vverbose  ]:    Very verbose output. Writes additional progress information in the standard output.
[    --vista     ]:        Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[ -m --memory    ]:    Approximate RAM memory amount to be made available and used by the program (in GBytes).
                           Valid values [0.1,50]. Default: 0.5.
[ -z --zip       ]:    Zip output files.
[    --nolog     ]:    Treat input tractograms as being normalized in natural units rather than logarithmic.
[ -p --pthreads  ]:    Number of processing threads to run the program in parallel. Default: use all available processors.
```

* Usage example:

```
distmatrix -r roi_lh.txt -I tracograms/ -O results/ -t 0.001 -b 5000 -v -m 5 -z
```

* Outputs (in output folder defined at option -O):

- '*roi_index.txt*': A file containing an index matching each seed coordinate to a block number and position within the block.
- '*dist_block_X_Y.nii(.v)*': Files containing the distance values for the submatrix in position XY within the full distance matrix.
- '*distmatrix_log.txt*': A text log file containing the parameter details and in-run and completion information of the program.

- ## buildgraphtree

Build a graph linkage hierarchical tree from a distance matrix built with *distmatrix*.

*** Arguments:**

```
      --version    :   Program version.
  -h  --help       :   Produce extended program help message.
  -r  --roi        :   A text file with the seed voxel coordinates and the corresponding tractogram index
                          (if tractogram naming is based on index rather than coordinates).
  -g  --graph      :   The graph linkage method to recalculate distances, use:
                          0=single, 1=complete, 2=average, 3=weighted, 4=ward(not verified).
  -I  --inputf     :   Input data folder (containing the distance blocks).
  -O  --outputf    :   Output folder where tree files will be written.
[ -v  --verbose   ]:   Verbose output (recommended).
[     --vista     ]:   Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[     --debugout  ]:   Write additional detailed outputs meant to be used for debugging.
[ -p  --pthreads  ]:   Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
buildgraphtree -r roi_lh.txt -g 2 -I distblocks/ -O results/ -v
```

*** Outputs (in output folder defined at option -O):**

- *'LINKAGE.txt'*: Contains the output hierarchical tree.
  (where LINKAGE is defines the method in option -g: single/complete/average/weighgted/ward)
- *'buildgraphtree_log.txt'*: A text log file containing the parameter details and in-run and completion information of the program.

[extra outputs when using --debugout option]

- *'LINKAGE_debug.txt'*: Tree file with redundant information for debugging purposes.

- ## cpcc

Compute the cophenetic correlation coefficient (Farris, 1969) of a hierarchical tree.

*** Arguments:**

```
      --version    :   Program version.
  -h  --help       :   produce extended program help message.
  -t  --tree       :   File with the hierarchical tree to compute cpcc from.
  -I  --inputf     :   Input data folder containing the blocks of the precomputed tract pairwise distance matrix.
[ -v  --verbose   ]:   Verbose output (recommended).
[     --vista     ]:   Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[ -p  --pthreads  ]:   Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
cpcc -t tree_lh.txt -I distBlocks/ -v
```

*** Outputs:**

- Introduces the cpcc value in the #cpcc field of the tree file defined at option -t.

- **randtracts**

Generates a set of tractograms matching in number and name to those of an existing roi file but that will generate a randomly uniform dissimilarity matrix when computing their distance. This is intended to be able to establish a baseline for tree quality values, by testing the quality measures on trees built over tractograms with no similarity structure (random). For this purpose the tractograms must be vectors uniformly distributed on the positive quadrant surface of the unit hypersphere.

The main method achieves this by first generating normally distributed vectors in the hyperspace, then normalizing these vectors to the unit hypersphere. However, for this approximation to be accurate dimension must be relatively low, therefore a dimension=10 is recommended (default).

The alternative method generates uniformly distributed vectors in the hyperspace, then filters out elements outside the unit hypersphere, and normalizes the remaining elements to the surface. This method has higher accuracy than the main method at higher dimensions, but computing time also increases exponentially, as most elements must be filtered out.

*** Arguments:**

```
     --version   :   Program version.
  -h --help      :   Produce extended program help message.
  -r --roi       :   Roi file with leaf coordinates/trackIDs of tractograms to generate.
  -O --outputf   :   Output folder where results be written.
[ -d --dim      ]:   Desired dimension of the random tractograms. Default: 10.
[ -s --seeds    ]:   Random number generator seed. Change in order to obtain a different set of results.
                        Same seed will always be reproducible. Default: 0.
[    --alt      ]:   Use alternative method: filtered uniform hyperspace vector.
                        Better approximation at higher dimension values but much more time-consuming.
[ -v --verbose  ]:   Verbose output (recommended).
[    --vista    ]:   Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files.
[ -z --zip      ]:   Zip output files.
[ -F --ufloat   ]:   Use float32 representation to write output tracts (default is uint8).
[ -p --pthreads ]:   Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
randtracts -r roi.txt -O results/ -d 10 -s 0 -v
```

*** Outputs (in output folder defined at option -O):**

- *'probtract_X.cmpct'*: (default) - (where X is a tract ID) artificial compact tractograms that would yield a uniformly random distance matrix.
- *'connect_X_Y_Z.v'*: (--vista option) - (where XYZ are tract seed voxel coordinates) artificial compact tractograms in vista format that would yield a uniformly random distance matrix.
- *'randtracts_log.txt'*: A text log file containing the parameter details and in-run and completion information of the program.

• buildrandctree

Build a centroid hierarchical tree from a set of artificially pre-generated set of tractograms yielding a uniformly random similarity matrix and a seed neighbourhood information voxel list.

*** Arguments:**

```
     --version    :   Program version.
  -h --help       :   Produce extended program help message.
  -r --roi        :   A text file with the seed voxel coordinates and the corresponding tractogram index
                         (if tractogram naming is based on index rather than coordinates).
  -I --inputf     :   Input data folder (containing the compact tractograms).
  -O --outputf    :   Output folder where tree files will be written.
[ -d --maxnbdist ]:   Maximum dissimilarity a seed voxel tract must have to its most similar neighbour not be discarded.
                         Valid values: (0,1] Use a value of 1 (default) if no discarding is desired.
[ -c --cnbhood   ]:   Use centroid method with C neighbourhood level. Valid values: 6, 18, 24(default), 32, 96, 124.
[ -S --basesize  ]:   Merge homogeneous base nodes of size S. (mutually exclusive with -N option).
                         Default: 0 (no homogeneous merging).
[ -N --basenum   ]:   Grow N homogeneous base nodes. (mutually exclusive with -S option).
                         Default: 0 (no homogeneous merging).
[ -v --verbose   ]:   Verbose output (recommended).
[    --vista     ]:   Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[ -m --cache-mem]:    Maximum amount of RAM memory (in GBytes) to use for temporal tractogram cache storing.
                         Valid values [0.1,50]. Default: 0.5.
[ -k --keep-disc ]:   Keep discarded voxel information in a specialised section of the tree.
[    --debugout  ]:   Write additional detailed outputs meant to be used for debugging.
[ -p --pthreads  ]:   Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
buildrandctree -r roi_lh.txt -I tractograms/ -O results/ -c 26 -N 1000 -k -m 2 -v
```

*** Outputs (in output folder defined at option -O):**

- '*cX_bin_nmt.txt*': (where X is the neighbourhood level defined at option -c) non-monotonic binary-branching hierarchical tree without tree processing (if desired use *processtree* command).
- '*baselist_nmt.txt*': Meta-leaves (base nodes defined by the us of option -N or -S) list with IDs corresponding to the non-monotonic tree file.
- '*success.txt*': An empty file created when the program has successfully exited after completion (to help for automatic re-running scripting after failure).
- '*buildrandtree_log.txt*': A text log file containing the parameter details and in-run and completion information of the program.

[extra outputs when using --debugout option]

- '*cX_bin_nmt_debug.txt*': version of the counterpart file without '_debug' suffix with redundant information for debugging purposes.

- **processtree**

Do tree processing, either full raw tree preprocessing (monotonicity correction, base-node flattening, debinarization...) or/and linear node collapse. For more information on the preprocessing steps refer to (Moreno-Dominguez, 2014).

For an interactive tree processing management with more options please use the *Hierarchical Clustering* module developed in *OpenWalnut* (www.openwalnut.org).

*** Arguments:**

```
     --version     :   Program version.
  -h --help        :   Produce extended program help message.
  -t --tree        :   File with the hierarchical tree to preprocess.
  -O --outputf     :   Output folder where processed tree files will be written.
[ -n --name       ]:   Prefix for the output tree filename.
[ -c --collapse   ]:   Perform linear node collapse in order to de-binarize non-binary structures.
                          Recommended collapse factor value: 0.05.
[ --ignorebases   ]:   (use only alongside option -c) ignore node-base status when performing node collapse.
[ -r --raw        ]:   Do full processing of binary raw input tree.
[ -b --bases      ]:   (use only alongside option -r) do base-nodes (meta-leaves) flattening.
                          Requires file with base-nodes identifiers.
[ -m --monmult    ]:   Monotonicity error multiplier. Increase if monotonicity correction enters an infinite loop.
                          Default value: 1 (no multiplier).
[ -v --verbose    ]:   Verbose output (recommended).
[    --vista      ]:   Write output tree in vista coordinates (default is nifti).
[    --debugout   ]:   Write additional detailed outputs meant to be used for debugging.
```

*** Usage example:**

```
processtree -t tree_lh.txt -O results/ -n processedtree -raw -c -v
```

*** Outputs (in output folder defined at option -O):**

- The processed tree file with either the same original name as the one defined by option -t, or the name defined by option -n when used.
- If both option -r and -c are used, the previous statement refers to the file with the processed raw-tree and the furthermore collapse output will be written with the '_collapsed'' suffix.
- *'processtree_log.txt'* - A text log file containing the parameter details and in-run and completion information of the program.

- ## partitiontree

Obtain tree partitions at all granularity levels using the Spread-Separation method (finding the partition with highest SS index at each granularity). Optimal SS value for each partition is searched within a defined search-depth number of hierarchical levels. Final partitions can be filtered with a defined kernel size to keep local SS maxima within that kernel. For SS index refer to (Moreno-Dominguez, 2014)

For an interactive 3D partition management with more options please use the *Hierarchical Clustering* module developed in *OpenWalnut* (www.openwalnut.org).

* Arguments:

```
     --version       :    Program version.
  -h --help          :    Produce extended program help message.
  -t --tree          :    File with the hierarchical tree to extract partitions from.
  -O --outputf       :    Output folder where partition files will be written.
[ -d --search-depth]:    Search optimal partition for each granularity within d hierarchical levels.
                            A higher value will produce more optimized partition but will increase computing time.
                            Default: 3. Recommended values: 3 for good quality and fast computation, 4 for enhanced quality.
[ -r --filter-radius]:   Filter output partitions to keep only local SS (partition quality) maxima within an r-sized kernel across the
                            granularity dimension.
[ -h --hoz         ]:    Write horizontal cut partitions instead of SS ones (optimal partition search is still based on SS index).
[ -m --maxgran     ]:    Compute and write only the maximum granularity (meta-leaves) partition.
[ -v --verbose     ]:    Verbose output (recommended).
[    --vista       ]:    Write output tree in vista coordinates (default is nifti).
[ -p --pthreads    ]:    Number of processing threads to run the program in parallel. Default: use all available processors.
```

* Usage example:

```
partitiontree -t tree_lh.txt -O results/ -d 3 -r 50 -v
```

* Outputs (in output folder defined at option -O):

<u>Default outputs</u>
- *'allSSparts_dX.txt'*: (where X is the search depth level defined at parameter -d) Contains a summary of the partition information (cut value and size) for all granularities.
- *'TREE_SSparts_dX.txt'*: (where TREE is the filename of the input tree defined at parameter -t) contains a copy of the original tree file with the partitions at all granularities included in the relevant fields.
- *'partitiontree_log.txt'*: A text log file containing the parameter details and in-run and completion information of the program.

<u>Additional if using option -r</u>
- *'filtSSparts_dX_rY.txt'*: (where Y is the filter radius defined at parameter -r) Contains a summary of the resulting filtered partitions.
- *'TREE_SSparts_dX_rY.txt'*: Contains a copy of the original tree file with the resulting filtered partitions included in the relevant fields.

(when using --hoz option, the prefix 'SS' will be replaced by 'Hoz'')

<u>Alternative outputs when using option --maxgran</u>
- *'fmaxgranPart.txt'*: Contains the size information of the resulting maximal granularity partition for that tree.
- *'TREE_maxgranPart.txt'*: Contains a copy of the original tree file with the resulting max granularity partition included in the relevant fields.

• treetracts

Compute the mean tractograms from hierarchical tree nodes and the original leaf tracts and write them in compact form.

*** Arguments:**

```
     --version     :   Program version.
  -h --help        :   Produce extended program help message.
  -t --tree        :   File with the hierarchical tree to compute node tractograms from.
  -I --inputf      :   Input data folder (containing the seed voxel compact tractograms).
  -O --outputf     :   Output folder where tractogram files will be written.
[ -n --nodes      ]:   Write tracts for the following node ids (separated with whitespaces).
[ -b --bases      ]:   Write only the tracts corresponding to the base-nodes (meta-leaves).
[ -a --all        ]:   Write tracts for all the tree nodes.
[ -f --full       ]:   Write full 3D image tracts instead of compact tracts, indicate location of wm mask file here.
[ -c --clustmsk   ]:  Write for each tract the corresponding 3D mask of all the seed voxels contained in the corresponding cluster.
                           Indicate location of wm mask file here.
[    --notracts   ]:   [use only with -c] Do not write tracts to file (only cluster masks).
[ -v --verbose]:       Verbose output (recommended).
[    --vista]:         Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[ -m --cache-mem]:     Maximum amount of RAM memory (in GBytes) to use for temporal tractogram cache storing.
                           Valid values [0.1,50]. Default: 0.5.
[ -z --zip        ]:    Zip output files.
[ -F --ufloat     ]:   Use float32 representation to write output tracts (default is uint8).
[    --debugout   ]:   Write additional detailed outputs meant to be used for debugging.
[ -p --pthreads   ]:   Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
treetracts -t tree_lh.txt -I tracograms/ -O results/ -n 40 65 -b -m 2 -v -c
```

*** Outputs (in output folder defined at option -O):**

- '*compact_X.cmpct(.v)*': (where X is the corresponding node ID) A compact tractogram with the mean tractogram corresponding to X node cluster.
- '*fulltract_X.nii(.v)*': (when using -f option) A full 3D tractogram with the mean tractogram corresponding to X node cluster.
- '*cluster_X.nii(.v)*': (when using -c option) A full 3D mask with the seed voxels corresponding to X node cluster.
- '*treetracts_log.txt*': A text log file containing the parameter details and in-run and completion information of the program.

• comparetrees

Matches leaves or meta-leaves (base-nodes) across trees and computes tree comparison values (*tCPCC* and *wTriples*).

* Arguments:

	--version	:	Program version.
-h	--help	:	Produce extended program help message.
	--cl	:	[xor with --cg and --cr] direct leaf-wise correspondence. Use for matching trees built over the same seed voxel tractograms. Due to expected high number of leaves and to reduce computing time triples will be subsampled by 1/LEAF_TRIPLES_FREQ(1/10) (to change this value modify at source code).
	--cg	:	[xor with --cl and --cr] Greedy-match base-node-wise correspondence, indicate file where to write/load base-node dissimilarity matrix. Matches with a dissimilarity higher than DISSIM_THRESHOLD(0.9) will not be considered a match (to change this value modify at source code).
	--cr	:	[xor with --cl and --cg] Random base-node-wise correspondence. Used to obtain a random chance baseline for tcpcc and triples value to compare to. RAND_REPEAT(100) repetitions will be computed and triples will be subsampled by 1/RAND_TRIPLES_FREQ(1/3) (to change this value modify at source code).
	--t1	:	File with first tree to be matched and compared.
	--t2	:	File with second tree to be matched and compared.
	--f1	:	Folder with the tracts for the first tree. If --cl is chosen the folder should contain leaf tracts. If --cg or --cr options are chosen, it should contain base-node tracts and cluster masks warped to a common space.
	--f2	:	Folder with the tracts for the second tree. If --cl is chosen the folder should contain leaf tracts. If --cg or --cr options are chosen, it should contain base-node tracts and cluster masks warped to a common space.
-O	--outputf	:	Output folder where result files will be written.
[-t	--threshold	]:	Number of streamlines relative to the total generated that must pass through a tract voxel to be considered for tract similarity (i.e.: minimum value of a normalized probabilistic tract in natural units to be considered above noise). Valid values: [0,1) Use a value of 0 (default) if no thresholding is desired.
[-d	--eucdist	]:	Maximum Euclidean distance (in number of isotropic voxel distance units) between matched base-node cluster centre coordinates to be accepted as a valid match. Base-nodes considered for match with a higher Euclidean distance (in common space) will be considered without match if no better matching possibilities exist. [use only with --cg or --cr] Default: 20 voxel distance units.
[-n	--noise	]:	[use only with --cg] matching-noise correction. insert alpha value (0,1]. Matching noise will not take into account for comparison any tree structure below the noise level. The noise level for a given node in the tree is computed as the average matching distance of the contained base nodes multiplied by a linear alpha coefficient to control noise weighting. An alpha value of 0 will compute results at the full [0,1] alpha value range at 0.05 intervals. Refer to (Moreno-Dominguez, 2014) for more information on the matching-noise scheme.
[	--nocomp	]:	Only obtain tree correspondence, not the trree comparison values (tcpcc nor triples). Ignored if in --cr mode.
[	--notriples	]:	Only obtain the correspondence and tcpcc value, not the triples (the latter is significantly more time-consuming).
[-v	--verbose	]:	Verbose output (recommended).
[	--vista	]:	Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[-p	--pthreads	]:	Number of processing threads to run the program in parallel. Default: use all available processors.

* Usage example:

```
comparetrees -cg distMatrix.nii --t1 tree1.txt --t2 tree2.txt --f1 tracts1/ --f2 tracts2/ -O results/ -t 0.001 -d 20 -v
```

* Outputs (in output folder defined at option -O):

<u>Common outputs when using --cl option</u>

- 'compactValues.txt':	File containing the main tree-comparison values: tCPCC and wTriples (if not using --notriples option)
- 'compValues.txt':	File with extended comparison outputs and tree-matching quality values.
- 'treeCompared1.txt':	Tree file with the final first that was compared (after further processing at the matching stage).
- 'treeCompared2.txt':	Tree file with the final second tree that was compared (after further processing at the matching stage).
- 'comparetrees_log.txt':	A text log file containing the parameter details and in-run and completion information of the program.

<u>Additional outputs when using -cg option</u>

- MATRIXFILE:	(with filename indicated at option --cg) - The 2D matrix file with the basenode-tract distances across trees.
- 'protoCorrespTable.txt':	A file with the base node correspondence across trees before final unmatched-nodes elimination (original trees).
- 'finalCorrespTable.txt':	A file with the base node correspondence across trees after final unmatched-nodes elimination (final trees).

<u>Outputs when using -cr option</u>

- 'randCpct.txt':	File with the tCPCC outputs for the randomized matching runs.
- 'randStriples.txt':	File with the wTriples outputs for the randomized matching runs.

- ## fliptracts

Compute the mean tractograms from hierarchical tree nodes and the original leaf tracts and write them in compact form. Options are: flip all leaf tracts defined in a roi file or flip the desired node tracts by ID or as defined in a tree file.

*** Arguments:**

```
      --version     :   Program version.
  -h  --help        :   Produce extended program help message.
  -m  --mask        :   White matter mask image that was used to compact the tracts.
[ -r  --roi        ]:   [xor with -t and -n] Roi file with leaf coordinates/trackIDs of tractograms to flip.
[ -t  --treebases  ]:   [xor with -r] Hierarchical tree with base-node ids of precomputed tractograms to flip.
[ -a  --all        ]:   [use only alongside -t] When used, all precomputed tree-node corresponding tractograms will be flipped.
[ -n  --nodes      ]:   [xor with -r] A sequence of ids of precomputed node-tractograms to flip.
  -I  --inputf      :   Folder with the input tractograms to be flipped.
                           These should be leaf tracts for option -r and node tracts for options -t and -n.
  -O  --outputf     :   Output folder where results be written.
[ -o  --no-ow      ]:   When used, tracts will NOT be overwritten if the corresponding output file already exists.
[ -v  --verbose    ]:   Verbose output (recommended).
[     --vista      ]:   Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files.
[ -z  --zip        ]:   Zip output files.
[ -F  --ufloat     ]:   Use float32 representation to write output tracts (default is uint8).
[ -p  --pthreads   ]:   Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
fliptracts -m wmask.nii -t tree.txt -n 20 234 -I meantracts/ -O results/ -v
```

*** Outputs (in output folder defined at option -O):**

- *'compact_X.cmpct(.v)'*: (where X is the corresp. node ID): A compact tractogram with the X-flipped tractogram for X node cluster.
- *'fliptracts_log.txt'*: A text log file containing the parameter details and in-run and completion information of the program.

- ## fliptree

Flips the seed voxel coordinates saved in a tree file in the x-dimension (use to compare trees across hemispheres).

*** Arguments:**

```
      --version     :   Program version.
  -h  --help        :   Produce extended program help message.
  -t  --tree        :   File with the hierarchical tree to flip voxel coordinates from.
  -O  --outputf     :   Output folder where the x-flipped tree file will be written.
[ -v  --verbose    ]:   Verbose output (recommended).
[     --vista      ]:   Write output tree in vista coordinates (default is nifti).
```

*** Usage example:**

```
fliptree -t tree_lh.txt -O results/ -v
```

*** Outputs (in output folder defined at option -O):**

- *'TREE_flipX.txt'*: (where TREE is the tree filename defined at option -t) Contains the output X-flipped hierarchical tree.
- *'fliptree_log.txt'*: A text log file containing the parameter details and in-run and completion information of the program.

- **matchpartition**

Finds the best matching corresponding partitions in a target tree to those present in an unrelated reference tree (meta-leaf matching across these two trees must have been precomputed using comparetrees).

Two partition matching algorithms are available: signature matching and overlap matching. Found target partitions will be colour-matched as best as possible. There is also the possibility of only colour-matching predefined partitions of the target tree to predefined partitions of the reference tree.

*** Arguments:**

--version :	Program version.
-h --help :	produce extended program help message.
-r --reference :	The tree file with the reference partitioned tree.
-t --target :	The tree file with the target tree to find matching partitions in (or with partitions to be colour-matched).
-m --leafmatch :	File with the meta-leaf matching information across both trees (output of comparetrees command).
-O --outputf :	Output folder where partitioned/color matched tree files will be written.
[-s --signature]:	Signature-based partition matching, insert lambda coefficient value. [xor with -o and -c]. In this method a pair signature matrices are computed for each reference-target partitions to find the quality of the match. Each signature matrix defines a value for each pair of base-nodes of the tree it belongs to: 1 the base nodes are found in the same cluster, 0 if otherwise. The higher the correlation between the reference and target-derived matrices, the best match is the target tree partition to the reference tree one. A smart hierarchical search through possible partitions is conducted to find the one with best signature matching. The lambda coefficient determines if and how a similar number of clusters in both partitions affect the matching quality value. Lambda=0 -> cluster number does not affect the quality value. Lambda=1 -> cluster value similarity has as much weight as signature correlation.
[-o --overlap]:	Overlap-based partition matching. [xor with -o and -c]. A match between two partitions is found by iteratively matching clusters with higher base-node overlap and resolving possible ambiguities. The matching quality between partitions is defined as the number of base-nodes pairs that are classified in the same way in both partitions (both in the same cluster or both in different clusters) against the total number of pair combinations. A smart hierarchical search through possible partitions is conducted to find the one with best signature matching.
[-d --depth]:	Partition search depth (for signature and overlap matching. A higher value will mean a more exhaustive search of the possible partitions, but also a higher computation time, especially if the partition to be matched has a high number of clusters (>100). The default value (0, recommended) will adaptively give high search depth to low-cluster partitions and lower search depth to high-cluster partitions.
[-c --justcolor]:	Perform only colour matching across reference and target tree partitions (both trees need to have the same number of precomputed partitions). In multiple-to-one matching cases clusters from the reference tree might also be recoloured to better identify matching relationships across partitions.
[-x --excl]:	Colour exclusively clusters that have a match, clusters without match will be recoloured white (on both reference and target trees)
[-v --verbose]:	Verbose output (recommended).
[--vista]:	Write output tree files in vista coordinates (default is nifti).

*** Usage example:**

```
matchpartition -r refTree.txt -t targetTree.txt -m matching.txt -O results/ -s 0.5 –v
```

*** Outputs (in output folder defined at option -O):**

- Partition-matched tree files.

- ## tractdist

Compute the distance (dissimilarity) value between two tractograms read from file.

*** Arguments:**

```
     --version     :   Program version.
  -h --help        :   Produce extended program help message.
  -a --tracta      :   Filename of first tractogram.
  -b --tractb      :   Filename of second tractogram.
[ -t --threshold ]:    Threshold to apply directly to the tractogram values before computing the dissimilarity (in order to avoid
                           tractography noise affect the result). Unlike in other hClustering commands, this threshold value is an
                           absolute value to apply to the tractogram data as is, not a relative threshold.
                           Valid values: [0,1) Use a value of 0 (default) if no thresholding is desired.
[    --vista     ]:    Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
```

*** Usage example:**

```
tractdist -a tractA.nii -b tractB.nii -t 0.2
```

*** Outputs:**

- Results are displayed on standard output (screen).

- ## pairdist

Retrieve the distance (dissimilarity) value between two tree leaves or nodes as encoded in the corresponding hierarchical tree. Additionally, distance between leaves can be retrieved from a distance matrix, and those from leaves/nodes computed directly from leaf/node tractograms.

*** Arguments:**

```
     --version     :   Program version.
  -h --help        :   Produce extended program help message.
  -t --tree        :   File with the hierarchical tree.
  -i --IDs         :   Input node IDs to compute the distance from, insert a pair of values, one for each node ID.
[ -l --leaves    ]:    Interpret input node IDs as leaf IDs.
[ -T --threshold ]:    Threshold to apply directly to the tractogram values before computing the dissimilarity (in order to avoid
                           tractography noise affect the result). Unlike in other hClustering commands, this threshold value is an
                           absolute value to apply to the tractogram data as is, not a relative threshold.
                           Valid values: [0,1) Use a value of 0 (default) if no thresholding is desired.
  -L --leaftractf  :   Folder with the leaf seed voxel probabilistic tracts. Will trigger direct computation of tractogram distance
                           (and prior computation of mean tractograms in case of node IDS). Tracts must be normalized.
  -N --nodetractf :    Folder with the node mean tracts. Tracts must be normalized. Do not use together with -leaves option.
  -M --matrixf     :   Folder with the dissimilarity matrix files. Use only together with -l option.
[    --vista     ]:    Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
```

*** Usage example:**

```
pairdist -t tree.txt -i 234 368 -T 0.4 -L leaftracts/ -N nodetracts/ -M matrix/
```

*** Outputs:**

- Results are displayed on standard output (screen).

- ## full2compact

Transform a full 3D Image probabilistic tractogram into a 1D compact tract vector.

*** Arguments:**

```
       --version    :   Program version.
   -h  --help       :   Produce extended program help message.
   -i  --input      :   [mutually exclusive with -f] Input full 3D image tractogram to be compacted, multiple inputs allowed separated by spaces.
   -f  --filenames  :   [mutually exclusive with -i] Text file with a list of multiple input filenames.
   -m  --mask       :   White matter mask image that was used to perform the tracking.
[  -o  --output     ]:  Output file or folder to write compact tract with no normalization.
[  -n  --nat-norm   ]:  Output file or folder to write compact tract with natural normalization (linear from 0 to 1).
                            By default output files will have _nat suffix, to avoid this use --nosuffix option.
[  -l  --log-norm   ]:  Output file or folder to write compact tract with logarithmic normalization (base 10 log + linear from 0 to 1).
                            By default output files will have _log suffix, to avoid this use --nosuffix option.
[  -s  --streams    ]:  [mandatory with the use of -n and/or -l options] The number of streamlines that were generated for each seed voxel to
                            obtain the probabilistic tracts.
[      --vista      ]:  Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[      --nosuffix   ]:  Do not add _nat nor _log suffix for normalized tract outputs
                            (different output folders should be chosen for each or they will be overwritten).
[  -z  --zip        ]:  Zip output files.
[  -F  --ufloat     ]:  Use float32 representation to write output tracts (default is uint8).
[  -p  --pthreads   ]:  Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
full2compact -i fulltract1.nii fulltract2.nii -m wm_mask.nii -o nonorm/ -n natnorm/ -l lognorm/ -s 5000 --nosuffix
```

*** Outputs:**

- When using -f option and/or the inputs defined by -o/-n/-l options are folder names:
  Output files will be written in the output folder defined by -o/-n/-l options, filenames will be the same as the original ones, and will have '_log' suffix in the case of -l option, and a '_nat' for -n option. To avoid the adding of this suffixes and making the output filenames of -l/-n options equal to the original filenames, add --nosuffix option to the command line.
- When using -i option and the inputs defined by -o/-n/-l are filenames:
  Output files will be written at the specific filename paths specified.

- ## compact2full

Transform a 1D compact tract vector into a full 3D Image tractogram.

*** Arguments:**

```
       --version    :   Program version.
   -h  --help       :   Produce extended program help message.
   -i  --input      :   [mutually exclusive with -f] Input compact tractogram to be blown into a full 3D image, multiple inputs allowed separated
                            by spaces.
   -f  --filenames  :   [mutually exclusive with -i] Text file with a list of multiple input filenames.
   -m  --mask       :   White matter mask image that was used to compact the tracts.
[  -o  --output     ]:  Output file or folder to write full tracts.
[      --vista      ]:  Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files].
[  -z  --zip        ]:  Zip output files.
[  -F  --ufloat     ]:  Use float32 representation to write output tracts (default is uint8).
[  -p  --pthreads   ]:  Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
compact2full -i compact_tract_1.nii compact_tract_2.nii -m wm_mask.nii -o output/
```

*** Outputs:**

- When using -f option and/or the input defined by -o option is a folder name:
  Output files will be written in the output folder defined by -o option, filenames will be the same as the original ones.
- When using -i option and the input defined by -o is a single filename:
  Output file will be written at the specific filename path specified.

• cmpct2vista

Transform a 1D .cmpct compact tract vector into a 1D .v vista image vector.

WARNING: this program will port the vector as-is, and is meant simply to be able to easily visualize the contents and facilitate conversion to other formats. A .cmpct vector compacted from nifti coordinates and then transformed to .v with this program, will not produce a correct image if then blown to full 3D with compact2full. To transform a .cmpct tract to a fully corresponding .v tract, firstly blow to 3D .nii image, then convert to vista with vnifti2image and then compact with full2compact.

*** Arguments:**

```
     --version   :  Program version.
  -h --help      :  Produce extended program help message.
  -i --input     :  [mutually exclusive with -f] Input .cmpct tractogram to be converted into vista 1D vector, multiple inputs
                       allowed separated by spaces.
  -f --filenames :  [mutually exclusive with -i] Text file with a list of multiple input filenames.
[ -o --output   ]:  Output file or folder to write vista 1D tracts.
[ -z --zip      ]:  Zip output files.
[ -F --ufloat   ]:  Use float32 representation to write output tracts (default is uint8).
[ -p --pthreads ]:  Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
cmpct2vista -i tract.cmpct -o tract.v
```

*** Outputs:**

- When using -f option and/or the input defined by -o option is a folder name:
  Output files will be written in the output folder defined by -o option, filenames will be the same as the original ones.
- When using -i option and the input defined by -o is a single filename:
  Output file will be written at the specific filename path specified.

• vista2cmpct

Transform a 1D .v vista image vector into a 1D .cmpct compact tract vector.

WARNING: this program will port the vector as-is, and is meant simply for completion of the cmpct2vista program. A .v vector compacted from vista coordinates and then transformed to .cmpct with this program, will not produce a correct image if then blown to full 3D .nii with compact2full. To transform a .v vector tract to a fully corresponding .nii tract, firstly blow to 3D .v image, then convert to nifti with vimage2nifti and then compact with full2compact.

*** Arguments:**

```
     --version   :  Program version.
  -h --help      :  Produce extended program help message.
  -i --input     :  [mutually exclusive with -f] Input vista 1D vector to be converted into .cmpct format, multiple inputs allowed separated
                       by spaces.
  -f --filenames :  [mutually exclusive with -i] Text file with a list of multiple input filenames.
[ -o --output   ]:  Output file or folder to write .cmpct tracts.
[ -z --zip      ]:  Zip output files.
[ -F --ufloat   ]:  Use float32 representation to write output tracts (default is uint8).
[ -p --pthreads ]:  Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
vista2cmpct -i tract.v -o tract.cmpct
```

*** Outputs:**

- When using -f option and/or the input defined by -o option is a folder name:
  Output files will be written in the output folder defined by -o option, filenames will be the same as the original ones.
- When using -i option and the input defined by -o is a single filename:
  Output file will be written at the specific filename path specified.

• image2tree

Creates a tree with base nodes matching those of an input tree file and structure matching that of an input single partition 3D image. It uses the partition and roi coordinates information in the 3D image to assign each base-node (meta-leaf) of a hierarchical tree to one of the partition clusters, then create a new tree with the same base nodes as the original but with only one partition in the hierarchical structure: the most similar to the one defined in the 3D partition label image. This single-partition tree can then be used to perform tree-comparison statistics between the original tree and the single-partition tree.

*** Arguments:**

```
        --version    :   Program version.
   -h   --help       :   Produce extended program help message.
   -t   --tree       :   File with the hierarchical tree to be used as base node template.
[  -b   --bases      ]:  File with the tree meta-leaves (base-nodes) identifiers. If omitted base nodes will be calculated from the
                             tree.
   -i   --image      :   File with the 3D partition label image that wishes to be projected into a tree with matching base nodes to
                             the input tree.
   -O   --outputf    :   Output folder where partition files will be written.
[  -v   -verbose     ]:  Verbose output (recommended).
[       --vista      ]:  Write output tree in vista coordinates (default is nifti).
```

*** Usage example:**

```
image2tree -t tree.txt -i partition.nii -O results/ -v
```

*** Outputs (in output folder defined at option -O):**

- *'partitionTree.txt'*:	A copy of the original tree file with the best-matched partitions to the 3D label file included in the relevant fields.
- *'success.txt'*:	An empty file created when the program has successfully exited after completion (to help for automatic re-running scripting after failure).
- *'image2tree_log.txt'*:	A text log file containing the parameter details and in-run and completion information of the program.

• surfprojection

Performs an interpolated projection from the roi seed voxels to the vertices of a (freesurfer) surface. Options include nearest neighbour, averaging, and Gaussian interpolation.

*** Arguments:**

```
       --version    :   Program version.
    -h --help       :   Produce extended program help message.
[   -k --kradius   ]:   Kernel radius (in voxel dimension units). Use 0 for nearest neighbour interpolation (default) and > 0 for
                            average interpolation.
[   -g --gauss     ]:   [use only with -k and radius > 0] Use Gaussian smoothing instead of average. Indicate full-width half-
                            maximum (in voxel dimension units).
    -r --roifile    :   File with the seed voxels coordinates.
    -s --surffile   :   File with the surface vertex coordinates.
    -I --inputf     :   Input tractogram folder (leaf tractograms).
    -O --outputf    :   Output folder where resulting tracts will be written.
[   -v --verbose   ]:   Verbose output (recommended).
[      --vista     ]:   Read/write vista (.v) files [default is nifti (.nii) and compact (.cmpct) files.
[   -z --zip       ]:   Zip output files.
[   -F --ufloat    ]:   Use float32 representation to write output tracts (default is uint8).
[   -p --pthreads  ]:   Number of processing threads to run the program in parallel. Default: use all available processors.
```

*** Usage example:**

```
surfprojection -k 6 -g 3 -r roi.txt -s surf.txt -I leaftracts/ -O output/ -v
```

*** Outputs (in output folder defined at option -O):**

- '*compact_X.cmpct(.v)*': (where X is the corresponding surface vertex ID): A compact tractogram with the mean tractogram projected to vertex X.
- '*surfprojection_log.txt*': A text log file containing the parameter details and in-run and completion information of the program.