Usage

./gistpp –i infile –op operation [-i2 infile2][-o outfile][-opt options]

Table 1. List of operations with a brief explanation of their function.

|  |  |
| --- | --- |
| Operation | Function |
| group | Groups dx file based on non zero voxel values. |
| sasa | Solvent Accessible Surface Area: creates a surface which represents the closest voxels water occupy compared to another surface |
| mult | Multiply two dx files together (voxel by voxel) |
| add | Add two dx files together (voxel by voxel) |
| sub | Subtract two dx files (voxel by voxel) |
| div | Divide two dix files (voxel by voxel) |
| addconst | Add a constant value to each voxel in a dx file |
| multconst | Multiply a constant value to each voxel in a dx file |
| contour | Process one dx file for whether it fits a criteria defined in regards to a second dx file |
| filter1 | Process a dx file for whether it fits a single criteria within its own voxels |
| filter2 | Process a dx file both by a criteria applied to its own voxels but also by a criteria applied to another dx file |
| sum | Sums all values in the dx file and prints a final sum and the average voxel quantity found in the file |
| defbp | Reads in a dx file and a pdb file. Using the heavy atoms present in the pdb file returns a dx file where only proximal voxels contain data |
| printpdb | Converts a dx file to pdb format. The output pdb file will contain voxel coordinates as hydrogen atoms in which the occupancy value corresponds to the original voxel data. |
| popstat | Reads the gist output file and prints onto the command line statistics related to a few values not normally printed as dx files |
| Vdw | Reads in a dx file containing occupancy information and a user specified vdw radius (1.2 A for oxygens) and returns a file with all voxels within that range flagged. |
| Makedx | Reads in the text file output of gist and a dx file from the same gist output and allows the user to specify any column in the text file to be written as its own dx file (for values typically not output by default gist) |

Each of these functions will be explained in terms of usage more carefully below, with detailed explanations regarding options related to them.

For help gistpp allows users to request further information regarding operations and options via:

./gistpp –operation

./gistpp –options

and full help via:

./gistpp –h (or simply ./gistpp)

**Group**

Example usage:

./gistpp -i infile.dx –op group

The grouping function will read in the desired dx file and begin creating groups around voxels which contain non-zero values. This is performed by searching through each voxel in order until a non-zero voxel is found, then recursively calling the same search on all neighbor voxels. The neighbors then, via recursion search their neighbors if they are also non-zero. This is done until no new group members are found (meaning all surrounding voxels are zero) or the heap is overloaded due to high group occupancy. To bypass heap overload groups cannot exceed voxel occupancy values of 10000.

Typical input files to this operation are dx files which have been previously processed and are in binary format (1 for data, 0 for empty) or a dx file type which contains 0 as a base case such as the g(X) file types.

The output of this function will be several files. A groupNN.dx file for each NN group which contains a dx file that only contains non-zero values in voxels which are within that group (the values stored will be the same voxel values that are present in the original input file). A grcount.txt file which lists each group by number and states their voxel occupancy. A mastergroup.dx and mastergroup.pdb file which contain location data for every group created during this process.

The benefit of the pdb output is the ability to visualize the file and identify groups of interest based on proximity to some feature of interest in the structure analyzed through gist. Specifically group number information is stored in the column within the pdb format reserved for resid.

**SASA**

Example usage:

./gistpp -i gist-gO.dx –op sasa -o SASA.dx

The SASA function must be run on a g distribution (typically desired to be used on oxygen distributions). This code searches through the dx file voxel by voxel until it finds a voxel with greater than 0.3. This value corresponds to an approximate cutoff representing water is present in this voxel during the simulation in question. Then each neighbor of that voxel is checked, if even one neighbor contains a value less than 0.1 the original voxel is flagged as being a part of this surface. The value 0.1 represents an approximate cutoff that represents a voxel that does not contain water throughout the simulation. Once all voxels have either been flagged as part of the surface or excluded a new dx file is printed in which only flagged voxels contain values (in this case the value will always be 1). This produced dx file when visualized will show the surface which is accessible to the solvent during the course of the simulation which is being analyzed.

**Mult, Add, Sub, Div**

Example usage:

./gistpp -i infile.dx -i2 infile2.dx [-op mult or add or sub or div] -o outfile.dx

These mathematical functions all proceed in the same manner. Each voxel in the first file is transformed by the function applied by the second file. Meaning voxels in file 1 will be multiplied by voxels in file 2 (or subtracted by). The resulting values for each voxel is then printed into an outfile.

**Addbyconst, Multbyconst**

Example usage:

./gistpp -i infile.dx [-op addconst or multconst] –opt const C -o outfile.dx

These functions will read in the dx file and add or multiply by the provided constant C each voxel value, then finally printing the new values for each voxel into the outfile.

**Contour**

Example usage:

./gistpp -i infile.dx -i2 infile2.dx –op contour –opt cutoff1 X [-opt gt1 or lt1] -o outfile.dx

This function reads in the first infile and either keeps the voxel values found within or sets them to 0 based on voxel data from infile 2. The options are as follows: cutoff1 allows the user to set a numerical value to assess against and gt1 relates to greater than and lt1 relates to less than. For example: ./dxpp -i1 infile.dx -i2 infile2.dx -contour -cutoff 0.5 -gt1 -o outfile.dx will produce an outfile with voxel values matching infile.dx where the voxels in infile2.dx are greater than 0.5.

This allows the user to investigate one dx file provided its voxels fit a different criteria. For example this could allow a user to investigate voxel energies only where high g(O) values exist.

**filter1**

Example usage:

./gistpp -i infile.dx –op filter1 –opt cutoff1 X [-opt gt1 or lt1] -o outfile.dx

This function will read in a dx file and set all voxel values to zero which do not fit a criteria set by the runtime options. Any voxel which does fit the criteria will have its value set to 1, meaning the output is binary, 1 means the voxel matches the expected value and 0 means it did not.

Example:

./gistpp -i gist-gO.dx –op clear1 –opt cutoff1 2.0 –opt gt1 -o highg.dx

In this example all voxels which have higher than 2 g(O) will be set to 1 and printed in the outfile highg.dx.

**filter2**

Example usage:

./gistpp -i infile.dx -i2 infile2.dx –op filter2 –opt cutoff1 X [-opt gt1 or lt1] –opt cutoff2 Y [-opt gt2 or lt2] -o outfile.dx

This function will read in dxfile 1 and set its voxel values to 0 or 1 dependent on whether its values match both the criteria in cutoff1 (applied to file 1) and cutoff2 (applied to file 2). The resulting outfile will contain 0's where the voxels did not match one or both criteria and 1's where both files voxels matched the expected criteria.

Example:

./gistpp -i gist-gO.dx -i2 Esw-dens.dx –op clear2 –opt cutoff1 2.0 –opt gt1 –opt cutoff2 0.5 –opt gt2 -o highghighe.dx

In this run the resulting file will have 1's in voxels which were found to have both high g values and high Esw values.

**Sum**

Example usage:

./gistpp -i infile.dx –op sum

This function will read in all voxel values in the infile and sum them. Then print that value along with the average per voxel quantity in the command line terminal.

**Defbp**

Example usage:

./gistpp –i infile.dx –i2 ligand.pdb –op defbp –opt cutoff1 X –o outfile.dx

\*Note the pdb format file must be entered in i2 NOT in i.

This operation will read in the coordinates from ligand.pdb and store all heavy atoms. Then each voxels position will be compared to the distance X from any of those heavy atoms, if the voxel is found to be within that distance its value will be set to 1, if not its value will be set to 0. The results will be written to outfile.dx. This file will contain a binary dx file definition of a defined region about all ligand.pdb heavy atoms.

This operation can be used to find a binding pocket (where it derives its name) or in combination with the sum and mult operations to integrate any values around a heavy atom of interest. The workflow would be to define the region with defbp, mult the resulting output with a dx file containing values, then sum the final output.

**Popstat**

Example usage:

./gistpp –i infile.out –op popstat

This operation will read in the gist output file and print in the command line various statistics related to voxel populations and norm values.

**Printpdb**

Example usage:

./gistpp –i infile.dx –op printpdb

This operation will read in the dx file and print a pdb file in which each voxel from the dx file will be a hydrogen atom with an occupancy value related to the original dx file voxel value.

**Vdw**

Example usage:

./gistpp –i infile.dx –op vdw –opt const ##

This operation reads in a dx file in which all voxels containing atom data are flagged, and a user specified vdw radius (-opt const ##) and searches for any voxels within the maximum vdw radius and flags them. This allows the end-user to see the maximum size of vdw interactions of a specified group of molecules.

**Makedx**

Example usage:

./gistpp –i infile.out –i2 infile.dx –op makedx –opt const ##

This operation outputs the data found in column ## from infile.out into a dx file with the same header information as infile.dx. Both infile’s must be from the same gist output. This operation allows end users to create visualizations of data not output by the default gist run, such as normalized quantities of energy and entropy.

Example output of ./gistpp –operation:

Following is a list of the available operations provided in this code

The format is as follows:

-op [operation][# infile][type infile][outfile][synopsis]

where:

operation is the command flag

#infile = number of infiles required (1 or 2)

type infile is the required infile format(dx, gist, or pdb where: gist is the gist outfile)

outfile is whether or not the code requires a specified outfile (Y/N)

synopsis is a brief explanation of the operation

-op group 1 dx N produces files which contain spatially grouped voxels with similar qualities

-op contour 2 dx Y produces an outfile which contains the values of the first provided the same voxel meets a criteria in the second file

-op clear1 1 dx Y produces an outfile which contains all 1's and 0's, 0 if the voxel does not fit a desired criteria in infile (ex: high energy), 1 if the voxel does

-op clear2 2 dx Y same as clear1 only requires the voxel to fit criteria in 2 dx files rather than simply 1 (ex: high energy and high g(O))

-op sasa 1 dx Y produces a dx file which contains 1's in each voxel which defines the solvent accessible surface area

-op sum 1 dx N prints in the command line the sum of all voxel quantities and the average voxel quanitity for the values found in the dx file provided

-op add 2 dx Y produces a dx file in which each voxel is the sum of the 2 corresponding voxels in each input file

-op sub 2 dx Y produces a dx file in which each voxel is the difference of the 2 corresponding voxels in each input file (file 1 - file 2)

-op div 2 dx Y produces a dx file in which each voxel is the quotient of the 2 corresponding voxels in each input file (file1/file2)

-op mult 2 dx Y produces a dx file in which each voxel is the product of the 2 corresponding voxels in each input file

-op addconst 1 dx Y produces a dx file in which each voxel is changed by the addition of a specified constant

-op multconst 1 dx Y produces a dx file in which each voxel is changed by the multiplication of a specified constant

-op defbp 2 dx+pdb Y produces a dx file in which each voxel is flagged as 1 when within a set distance of any ligand.pdb heavy atoms

-op printpdb 1 dx Y produces a pdb file containing hydrogen atoms at every voxel with the gist voxel data stored in occupancy

-op popstat 1 gist N prints in the command line statistics taken directly from the gist outfile

Following is an example of ./gistpp –options:

Following is a list of the available options and which operations they correspond to

The format is as follows:

-opt [option][##][operation][synopsis]

where:

option is the command flag

## indicates a numerical value must be added after the option

operation is the list of operations this option is required for

synopsis is a brief summary of the options effect

-opt cutoff1 ## [clear1 clear2 defbp] this option specifies the cutoff value which is desired to be applied to infile or the desired distance in defbp

-opt cutoff2 ## [clear2 contour] this option specifies the cutoff value which is desired to be applied to infile2

-opt gt1 [clear1 clear2] this option specifies we are interested in values greater than cutoff1 in infile

-opt lt1 [clear1 clear2] this option specifies we are interested in values less than cutoff1 in infile

-opt gt2 [clear2 contour] this option specifies we are interested in values greater than cutoff2 in infile2

-opt lt2 [clear2 contour] this option specifies we are interested in values less than cutoff2 in infile2

-opt const ## [addconst multconst] this option specifies the constant to add or multiply by in the provided function