GISA "Read me"

C. Grønbæk

July 19, 2019

1 Implementation notes

The GISA-algorithm and tools are implemented in C-code placed in a small library. Library content/versions:

GISA_v10_unix.c
GISA_main_v10_unix.c
SAonGISA_v10_unix.c
SAonGISA_main_v10_unix.c
dataTypeMatching_v3_unix.c
sortingUtils_v4_unix.c
mathUtils.c

All files and compiled versions GISA_main_v10_unix.o and SAonGISA_miain_v10_unix.o are available in the repository ceegeeCode/GISA on www.github.com. Currently this only works in unix; earlier versions ran also in Windows, and this may be resurrected.

There are two main code "blocks": GISA_v10_unix.c and SAonGISA_v10_unix.c. The core of GISA_v10_unix.c is the function for computing the invariants (computeGI); this also supports the restricted and the unrestricted search. Secondly, GISA_v10_unix.c contains a (major) function for deriving invariant values on sub-chains and pairs of sub-chains (computeGI_windows); this function is the generating the input for the rarity scans. These (rawRarity0,1 and 2) are the core content of SAonGISA_v10_unix.c.

The files of similar names, but with a "main" in them, GISA_main_v10_unix.c and SAonGISA_miain_v10_unix.c, are wrappers of the two main code blocks so as to provide the command line tools for the underlying functions. Thus,

these two files contain main functions (main in the C-code meaning) calling the functions in GISA_v10_unix.c and SAonGISA_v10_unix.c and the code for parsing the calls (this uses the C-library file argp for parsing command line arguments).

2 Documentation

The code is generally documented. For instance, computeGI is preceded by a text block in which a general description of the function and how it works is found, along with details of its input parameters. All other functions in the GISA_v10_unix.c code are utilities used in the key block; these utility functions are preceded by a short description of their purpose. The functions aimed for the searching are though preceded by a longer description and lengthier comments are found within these blocks (this goes e.g. for the functions aggrAndW_wClosedLoops, examineClosedLoops and writheSubChainPairs). Elsewhere, shorter comments are found within the code. Similarly, the key scan functions, rawRarity0, 1 and 2, in SAonGISA_v10_unix.c are preceded with a documentation block. Another source of documentation are the outlines (for all these four functions) found in the Github repository; each of these files contains a quite detailed outline of the code for the function in consideration (as the name of the file lets know).

Regarding the input parameters to the various functions callable from the command lines, the –help option can as just mentioned be called.

3 Compiling and running

For compiling GISA, gcc can be applied as in the following commands:

gcc GISA_main_v10_unix.c -o GISA_main_v10_unix.o GISA_v10_unix.c dataTypesMatching_v3_unix.c mathUtils.c -lm

gcc SAonGISA_main_v10_unix.c -o SAonGISA_main_v10_unix.o SAonGISA_v10_unix.c GISA_v10_unix.c sortingUtils_v4_unix.c dataTypesMatching_v3_unix.c mathUtils.c -lm

To run the code you execute the .o-files, e.g to run GISA you simply go GISA_main_v10_unix.o and add the necessary input and set parameters (options) as wanted. To get help simply call –help. This provides details on

options/input parameters standardized as for other functions implemented on your unix machine. In the repository you can find examples of commands for running the code so as to get output as that shown in the paper.

The state of the code is that of "written for research usage". In particular, in calls of the code GREAT CARE must be exerted upon providing paths to existing directories and meaningful and complete input parameters. While the code does contain some checks, it does NOT (by far) contain all assertions needed to check that the input is meaningful and that all files/directories exist.

3.1 Generalities on input and output

The key input is to the computations are PDB-files, which must be placed in directories for the purpose. The output generally consists in files to which the results have been written; the names of the output file are created inside the code of the key functions by means of concatenations of hardcoded strings and the provided input parameters (more below).

The main input to the computations are PDB-files, assumed to sit in a single directory possibly with sub-directories (reading this directory is probably the major difference between the version for Windows and that for UNIX). The function for loading the PDB-files reads in the structures as "chains"; if a structure is a multi-mer in the sense that it contains several PDB chain id's, each sub-structure is read in as a separate chain and assigned its PDB chain id (if the PDB chain id is blank, > is assigned). Chains are skipped if they contain segments that are regarded as unlikely long (the square of the allowed length is set by the global parameter stdRealSegLength), if they are shorter than a globally set length (strLengthCutOff) or longer than a set max-length (function parameter maxChainLength). The accuracy used is double precision. Throughout the code sub-chains are called windows.

As for the naming of the output files a few examples probably suffice. The files are .txt file. For example the file

Invariants_windowlgth_30_2_order_2_1_computeGI_windows_top8000_winCovType0.txt

contains the Gauss integrals (invariants) on windows (sub-chains) of length

30 covering each structure at a step size of 2; the invariants cover order 2 including the absolute value versions; the computations where done with computeGI_windows for the Kinemage top8000 set; windows were generated with the type 0 method for that purpose.

The file

Invariants_Pairs_windowlgth_30_2_order_2_1_computeGI_
windows_top8000_winCovType0_onlyDisjointPairs1.txt

contains the Gauss integrals on window pairs of length 30 covering each structure at a step size of 2 (these are then mutual GIs); the invariants' order is ≤ 2 and include the absolute value versions; the computations where done with computeGI_windows for the Kinemage top8000 set; windows were generated with the type 0 method for that purpose; only (mutual) GIs for pairs of disjoint windows are had.

The file

RarityScan1_ScoresPairs_windowslgth_20_4_order_2_1_top100_ top8000_1wins_
norm_5invs_2mmsPairs_20bins1_winCovType0_threshMut5.000000_pValuesIncl.txt

contains the results (scores of pairs) of a rar1 rarity scan; window length was 20 covering each structure at a step size of 4; the input invariants cover order 2 GIs including the absolute value versions; the Kinemage top100 set was queried against the Kinemage top8000 set; one window (pair) at a time was used for matching; the matching was based on 5 (5invs) normed invariants allowing 2 mismatches (2mms); GIs were divided in 20 bins of type 1 (equidistant bins); windows were generated with the type 0 method for that purpose; a threshold on the (mutual) GIs of 5 was applied.

Here follows notes on the usage of the code for the rarity scans, each of which consists in a call of SAonGISA_main_*.o in one of three flavours, rar0, 1 or 2.

3.2 Flavour: rar0

In rar0 rareness is decided by means of mutual writhe (possibly absolute value thereof, but signed by default). The usage is (see also the example runs in the github repository):

1. Create the back ground "data base" of the wanted Gauss numbers for pairs of windows: this is done by running the main function here (SAonGISA_main_*) in the makeDB flavour. This wraps a call to the GI_windows function of the GISA_**_unix-code. Must be run in "window pairs mode" (and) so that a file of mutual writhe values for all pairs of disjoint windows for each structure in the PDB-set is created. Set the order to 1; set windowing parameters as desired (e.g. windowLength 20, stepSize 4); use windowType 0.

2. Now run the rarity scan with rar0 in one of the two modes for scoring (option: u; see the main paper and the supplementary for an explanation of the methods). Use the –help option to get details on the available input parameters. Among these a threshold can be set so as to filter away all pairs with absolute mutual writhe lower than this cutoff level.

3. When the scan is completed, the user is asked if another query set should be scanned; by doing this a (re)loading of the back ground data base is avoided.

When the scoring of a query is based on the max mutual writhe found in the query (option u1 and default), a probability ("p-value") of the query relative to the back ground is generated directly. If though the scoring is based on the absolute mutual writhes (-u0), it is necessary to first run a scan of the data base against it self (i.e. the query set is the same set as the data base) and with option P set to 0. When done, general query sets can be scanned to obtain probabilities of the queries relative to the back ground (set option P to 1). See also the supplementary.

Final scores are written to a file, in decreasing order (corresponding to lowest p-value at the top).

3.3 Flavour: rar1

The scoring in rar1 is done by means of a "cross entropy" (see the main paper and the supplementary for the explanation). The usage is though quite similar to rar0's (see also the example runs in the github repository):

1. Create the data base: this is done by running the main function here (SAonGISA_main_*) in the makeDB flavour. This wraps a call to the GI_windows function of the GISA_**-code and must be run in "window pairs mode" (and) so that a file of mutual writhe values or,

more generally, for the desired Gauss Integrals, for all pairs of disjoint windows for each structure in the PDB-set is created. Set the order according to the set of invariants wanted (i.e. 1 if only the writhe and, possibly, the average crossing number are desired, else set the order to 2). As mentioned in the supplementary, it is recommended only to use order 1, since the mutuals of the higher order invariants are not (all) true mutuals, which make these inappropriate for the matching. Set windowing parameters as desired (e.g. windowLength 20, stepSize 4); use windowType 0.

2. Now run the rarity scan with rar1: the search method is aimed at detecting the frequency of data base window pairs with Gauss numbers in "epsilon-distance" to those of a given query pair (of windows); for a given query structure these frequencies are collected and a score is obtained by accumulation. The "epsilon" is set indirectly through binning of the invariant values. (See the supplementary for an explanation of the method.)

3. When the scan is completed, the user is asked if another query set should be scanned; by doing this a (re)loading of the back ground data base is avoided.

As with rar0 in mode "abs-mutual-writhe" (u0), to obtain the p-values corresponding to the obtained scores output, it is necessary to first run rar1 with query set = data base set and with rarityScorePValues_b = 0 (P0) and else leave the settings). When done, run rar1 for the desired query set now with rarityScorePValues_b = 1 (P1).

Also as in rar0, it is possible to set a threshold filtering away all pairs with absolute mutual GI lower than this cutoff level. For speed it is (very) desirable to set this threshold quite high; when the scan is only based on the writhe (and no other invariants) the threshold can be set to e.g. 10 (and leave out normalization of the invariant values by setting the normalize_b parameter at 0). When running with more invariants (e.g. also the average crossing number) the values must be normalized (will be done to get mean 0 and st dev 1) and the threshold must then be set differently (maybe to 0.5); to set an appropriate threshold may need experimenting a little (e.g. have a look at the actual normalized invariant values – these values can be written out).

Final scores are written to a file, in decreasing order (corresponding to lowest p-value at the top).

3.4 Flavour: rar2

The rar2 method is very similar to that of rar1, the data base is though here "dictionaried" based on the Gauss numbers of the windows and not the window pairs. Pairs matching is optional and based on the single window matches. It is (therefore) possible to run a scan either in single window mode or have the pairs considered too.

The usage of rar2 is again quite similar to the two others, though with a few differences (see also the examples runs in the github repository):

1. Create the data base: this is done by running the main function here (SAonGISA_main_*) in the makeDB flavour. This wraps a call to the GI_windows function of the GISA_**-code. To run the scan in "window pairs mode" a data base for the pairs of windows must be created (run SAonGISA_main_* in flavour makeDB); this must not only contain results for the disjoint windows, but rather all combinations (since the single window results will be taken from this pairs data base). To run in "single window mode" a data base for the single windows results must be created (as for the pairs by running SAonGISA_main_* in flavour makeDB, but here of course calling for having the single window values computed and stored). In short: in either case have the data base created for the desired Gauss Integrals invariants. For this set the order according to the set of invariants wanted (ie set order to 1 if only the writhe and average crossing number are desired, else 2); for the pairs-based scan it is recommended only to use order 1 (nrOfEntriesForPairs can be set to 1, while nrOfEntries can be left higher so as to allow to have higher order Gauss numbers included in the single windows matching). Set windowing parameters as desired (e.g. windowLength 20, stepSize 4).

2. Now run the rarity scan with rar2: the search method is aimed at detecting the frequency of data base windows (and, possibly, pairs) with Gauss numbers in "epsilon-distance" to those of a given query window (pairs); for a given query structure these frequencies are collected and a score is obtained by accumulation. The "epsilon" is set indirectly through binning of the invariant values. (See the supplementary for an explanation of the method.)

3. When the scan is completed, the user is asked if another query set should be scanned; by doing this a (re)loading of the back ground data base is avoided.

As with rar1 to obtain the p-values corresponding to the obtained scores output, it is necessary to first run rar2 matching the data base up against itself (with P0). When done run rar2 for the desired query set now with rarityScorePValues_b = 1 (P1). Also as with rar1, it is possible to set a threshold filtering out pairs with abs mutual value lower than this cutoff level (and the same comments on its level apply).

4 Python and Pymol extras

The set of code placed in the Github repository also includes the Python file

gi_Anomalies.py

and two accompanying Pymol scripts

gi_Anomalies_restrictedSearch.txt
gi_Anomalies_unrestrictedSearch.txt.

This part of the code enables the plotting done in the main paper and in the supplementary. Please see inside these files for their usage (all three have a preamble dedicated to that purpose).