Appendix B

GOFISH Documentation

Overview

GOFISH is an external minimizer for the Coulomb excitation analysis code GOSIA which implements Particle Swarm Optimization (PSO). PSO is a metaheuristic optimization scheme, meaning that it optimizes without regard to the gradient of the function being minimized. GOSIA minimizations often have many highly correlated parameters and local minima, which limits the effectiveness of gradient-based optimizers. This led to the desire to develop a non-gradient-based optimizer which could speed up and improve the analysis for Coulomb excitation experiments. Particle Swarm Optimization was chosen because it is highly robust to high-dimensional search spaces with many local optima, and because it can be very effectively parallelized to take advantage of high-performance computing. This appendix focuses on documentation of the GOFISH code. For more information about the use of the GOFISH code in this work, see chapter 4. The GOFISH code is hosted at `https://github.com/avamh672/gofish`.

GOFISH begins by initializing a number of "particles" randomly in the parameter space, and giving them random initial velocities in this space based on the user-defined bounds for each parameter. Reasonable bounds for all matrix elements, and particularly buffer state matrix elements, are important to achieving a good result. After the particles are initialized, GOSIA's **OP,POIN** is evaluated for each particle to compute the yields associated with that particle's matrix elements, and a particle χ^2 value is computed by comparing the simulated yields with the experimentally observed ones. The particles'

velocities are then updated based on equation 4.1. The default values of the inertial, cognitive, and social coefficients (0.7, 1.75, and 1.0, respectively) were selected based on the results of a hyperparameter search. The user may, however, specify alternative values for these and many other parameters in the config file. The topology used in GOFISH is adaptive; it starts with an *lbest* topology with 20 neighbors, and adds more connections as the number of iterations increases until a *gbest* topology is achieved. By default GOFISH performs 500 iterations of particle swarm optimization, though this number can also be specified by the user. When the maximum number of iterations is reached, it writes the best position it found and the χ^2 value associated with that position to an output file.

File Preparation

GOFISH runs with mostly standard GOSIA input files, with the exception of a few comments which must be added to make the file more easily parsed. Complete GOSIA input files executing the **OP,INTI** and **OP,POIN** commands should be generated. More information on how to generate the GOSIA input files can be found in the GOSIA user manual (38). If simultaneous beam-target minimization is to be done, input files are needed for both the beam and target nuclei. Note that in the case of simultaneous minimization, the files should still be prepared as GOSIA input files, not GOSIA2 input files. You will also need to include all of the other files required to run those GOSIA input files; it should be possible to run GOSIA on each of the input files in your working directory. Additionally, GOFISH requires the inclusion of a config file and a specially-formatted yields file (or files, in the case of simultaneous minimization) which list all of the transitions being simulated. Examples of all required files are included in the Github repository for GOFISH (33).

The GOSIA Input File

In order to reduce the number of parameters which must be specified by the user, GOFISH uses the gosiaManager class to read the normalizing transition and upper limit, the liter-

ature constraints, and the relative scaling factors (if applicable) from the GOSIA input file or files. By flagging the relevant lines with comments, we can greatly simplify the functions required to parse the input file.

The following six lines, all in the **OP,YIEL** section, must be commented in each **OP,POIN** input file:

NS1,NS2 - This line must be commented with "!NS". GOFISH is not currently set up to handle different upper limit ratios or normalizing transitions for different experiments, so the input file should be prepared with only a single instance of these parameters.

UPL - This line must be commented with "!UPL".

NBRA - This line must be commented with "!BR".

NL - This line must be commented with "!NL".

NDL - This line must be commented with "!DL".

NAMX - This line must be commented with "!ME".

.

Additionally, if you doing beam- (or target-) only minimization, you will need to comment each line containing the variable **YNRM** with "!SCL", even for experiments which use independent scaling. The normalizing experiment numbers are read from the **EXPT** section of the GOSIA input file, and do not require any comments.

The Yields File

In addition to the standard yields file read by GOSIA, GOFISH requires a yields file which lists the yields for every transition associated with the matrix elements in the

GOSIA input. This yields file should be in the same format as the yields file described in section 7.33.1 of the GOSIA user manual (38), with the following modifications:

This yields file must include unobserved transitions whose matrix elements are included in the GOSIA input, in addition to the observed transitions. For unobserved transitions, the counts and uncertainty should be recorded as 0. GOFISH will interpret this as an unobserved transition and include it in the χ^2 calculation only if the computed yield exceeds the user-defined upper limit.

Transitions must be listed in the order in which they appear in the **OP,POIN** output.

Unresolved doublets must be formatted as described in section 7.33.1 of the GOSIA user manual (38). They must be listed in the yields file in the place of the first of the two transitions to appear in the **OP,POIN** output. Additionally, using the same variable names as in the user manual, the first transition to appear in the **OP,POIN** output must be denoted by II1 and IF1, and the second by II2 and IF2. The sum of the two transitions' yields will be treated as a single observable when calculating the χ^2 in GOFISH, as they would be in GOSIA.

The Config File

GOFISH has many parameters which can be specified by the user in the config file which is passed to the program at startup. There are both mandatory parameters, which must be specified by the user or the program will raise an exception and terminate, and optional parameters, which have default values which may be overridden by the user. An example config file is shown in Figure B.24.

```
gosia = /mnt/home/hillma54/gosia/gosia
beamINTIinp = sn116_gos2.INTI.inp
beamPOINinp = sn116_gos2.inp
beamYields = sn116yields.yld
nBeamParams = 9
nThreads = 21
scratchDirectory = /mnt/scratch/hillma54/
simulMin = True
targetINTIinp = pt196_gos2.INTI.inp
targetPOINinp = pt196_gos2.inp
targetYields = pt196yields.yld
nTargetParams = 22
nParticles = 600
nIterations = 500
inertialCoeff = 0.7
cognitiveCoeff = 1.75
socialCoeff = 1.0
velocityStrategy = invert
boundaryStrategy = reflective
parallelAnnealing = True
annealinglayers = 5
annealingRates = 0.3,0.4,0.5,0.6,0.7
crossoverIterations = 100,200,300,400
~
```

Figure B.24: An example config file for GOFISH.

Mandatory Parameters

The following parameters must be included in the config file. The naming convention (beam and target) assumes inverse kinematics, but the program itself makes no such assumption and will work for standard kinematics.

gosia - The path to the GOSIA executable. Note that there is a slightly modified version of GOSIA included in the Github repository for GOFISH. This version of GOSIA should be used.

beamINTIinp - The path to a GOSIA input file for the projectile which executes **OP,INTI**.

beamPOINinp - The path to a GOSIA input file for the projectile which executes **OP,POIN**. Note that certain modifications must be made to this file, specified in the section on file preparation.

beamYields - The path to the GOSIA input file which contains the yields for the projectile nucleus. Note that this file must be formatted in a particular way, specified in the section on file preparation.

nBeamParams - Integer, must be set to the number of matrix elements to be minimized for the beam.

nThreads - Integer, must be set to the number of threads to be used by the program. Note that since this program uses mpi4py, the number of threads specified here must match the number passed to the message passing interface at runtime. GOFISH uses a single director thread and uses the remaining threads to evaluate candidate solutions. It is therefore recommended that the number of evaluator threads divides evenly into the number of particles.

scratchDirectory - The path to the folder where GOFISH will generate subdirectories for running GOSIA in parallel. This folder can be empty; GOFISH will create the required subdirectories and copy over the necessary files.

simulMin - Boolean, must be set to either True or False. If set to True, GOFISH will perform a simultaneous minimization of the projectile and target matrix elements. If set to False, GOFISH will perform the minimization only for one set of matrix elements. Note that if simulMin is True, there are four additional parameters for the target which must be included in the config file.

targetINTIinp - The path to a GOSIA input file for the target which executes **OP,INTI**. This input is required only if simulMin is True.

targetPOINinp - The path to a GOSIA input file for the target which executes **OP,POIN**. Note that certain modifications must be made to this file, specified in the section on file preparation. This input is required only if simulMin is True.

targetYields - The path to the GOSIA input file which contains the yields for the target nucleus. Note that this file must be formatted in a particular way, specified in the section on file preparation. This input is required only if simulMin is True.

nTargetParams - Integer, must be set to the number of matrix elements to be minimized for the target. This input is required only if simulMin is True.

Optional Parameters

The following parameters may be optionally included in the config file. If they are not included, they will be set to the default values specified below.

nParticles - Integer, set to the number of particles to be included in the optimization. The default value is 600.

nIterations - Integer, set to the number of iterations for the optimization. The default value is 500.

inertialCoeff - Float, set to the inertial coefficient for PSO. The default value is 0.7. This parameter is not used if parallelAnnealing is set to True.

cognitiveCoeff - Float, set to the cognitive coefficient for PSO. The default value is 1.75.

socialCoeff - Float, set to the social coefficient for PSO. The default value is 1.0.

velocityStrategy - String, sets the method for correcting the velocities of particles which leave the parameter boundaries. This parameter may be set to any strategy accepted for the PySwarms velocityHandler class. The default value is 'invert'.

boundaryStrategy - String, sets the method for correcting the positions of particles which leave the parameter boundaries. This parameter may be set to any strategy accepted for the PySwarms boundaryHandler class. The default value is 'reflective'.

parallelAnnealing - Boolean, enables the parallel annealing option described below. Note that this option is significantly more computationally expensive and will drastically increase the run time of the program. The default value is False.

annealingLayers - Integer, set to the number of layers to be used for parallel annealing. This parameter is not used if parallelAnnealing is set to False. The default value is 5.

annealingRates - List of floats, set to the different inertial coefficients to be used for the different layers in parallel annealing. This parameter is not used if parallelAnnealing is set to False. The default value is [0.3,0.4,0.5,0.6,0.7].

crossoverIterations - List of integers, set to the iteration numbers at which layers will be swapped during parallel annealing. This parameter is not used if parallelAnnealing is set to False. The default value is [100,200,300,400].

Running GOFISH

The GOFISH code is reliant on GOSIA for evaluating the candidate solutions generated by particle swarm optimization. Candidate solutions are evaluated by setting all matrix elements to the candidate values and then using GOSIA's **OP,POIN** to estimate the yields associated with those matrix elements. The yields are then compared to the experimental yields to generate a χ^2 value. In order for GOFISH to run, all of the required inputs for GOSIA must be present run directory, and all files must be formatted as described in the section on file preparation. Prior to running GOFISH, you will also need to run the GOSIA **OP,INTI** input file (or files, for simultaneous minimization) to generate the "corrected" yields file (rather than correct the computed yields for the effect of the spline integration, GOSIA applies a correction to the experimental yields). Like GOSIA, GOFISH uses the corrected yields values for comparisons with the output of **OP,POIN**.

If simulMin is set to False, you will also need to run the GOSIA **OP,POIN** input file once before starting GOFISH. In order to reproduce the way experiment scaling factors are computed in GOSIA, GOFISH requires the DSIG values for each experiment. The version of GOSIA in the GOFISH Github repository has been modified to write these values to file when they are calculated in **OP,POIN**. GOFISH then reads the values from the GOSIA output file.

GOFISH handles multithreading through the Python library mpi4py, which provides bindings for the Message Passing Interface (MPI) standard. To run GOFISH, execute the following command, replacing the variables in {} with the appropriate values:

mpirun -n {nThreads} python gofish.py {configFile} {batchNumber}

nThreads - The total number of threads to be used. This must match the number of threads specified in the config file. GOFISH reserves one thread as the director that executes the optimization. The other threads are used for evaluating the candidate solutions. It is recommended that the number of evaluator threads divides evenly into the number of particles, but it is not required.

configFile - The path to the config file. The format of this file is described in the section on file preparation.

batchNumber - An integer, used only for naming the subdirectories created by GOFISH. If you are running multiple instances of GOFISH simultaneously, they must have different batch numbers so that they do not overwrite each other's files.

Particle swarm optimization is a stochastic algorithm and convergence is not guaranteed. As such, it is advisable to run multiple instances of GOFISH simultaneously to improve the odds of finding the global minimum. In the work presented in this thesis, GOFISH optimizations were run in batches of 20. Our observed rate of convergence to the apparent global minimum is approximately 90%, so it is feasible to run with smaller batches if computationally limited. GOFISH is designed with batch submission in mind, and multiple instances of the code running in the same directory will not conflict with each other as long as the provided batch numbers are different. You must ensure that this is the case; if two instances of GOFISH are run with the same batch number at the same time, even if they are run from different directories, they may overwrite each other's

temporary files.

When the GOFISH program terminates, it will produce a file called particleSwarmResult_{batchNumber}.csv which contains the best fit χ^2 value and matrix elements. The matrix elements will be listed in the order they are declared in the GOSIA input files, with beam matrix elements listed before target ones. Because the GOFISH relies on the output of **OP,INTI** to correct the measured γ-ray yields, the result is likely to be inaccurate unless your initial guess of the matrix elements was very close to the optimal values. To account for this, we recommend running GOFISH several times in succession, each time using the best fit matrix elements from the previous fit for the next **OP,INTI** calculation. This process should be repeated until the solution stops changing significantly. In the analysis of the ^{112,116,120}Sn Coulomb excitation, this step was performed three to four times for each isotope.

Finally, it is recommended that solution polishing be performed on the final result. While gradient-based optimizers often struggle in search spaces with many local optima, they are quite good at fine-tuning the solutions produced by other algorithms. We recommend that you employ a gradient-based optimizer (either the one built into GOSIA or another external minimizer) to polish the solution by using the GOFISH result as the initial guess for the gradient-based optimization and allowing only the matrix elements tied to observables to vary. The polished solution should be very close to the GOFISH result; if it is not, you may need to run more iterations of the **OP,INTI** $\rightarrow$ GOFISH loop.

Checkpointing

The GOFISH code takes on the order of hours to run, even when utilizing a large amount of computational resources. To avoid the waste of both time and resources, GOFISH employs a checkpoint system which writes all necessary program information to a series of files at the end of each iteration. The files will be stored in a folder called checkpoint_{batchNumber} which will be located in the GOFISH run directory. When a new

instance of GOFISH is started, it will automatically check for the existence of a checkpoint directory with the correct batch number. If it exists, the program will read in the checkpointed information and continue the optimization from that point rather than beginning from scratch. If you wish to start a run from scratch, you must first ensure that no such checkpoint directory exists. The checkpointing system was developed specifically for running on the scavenger queue at Michigan State University's Institute for Cyber-Enabled Research (ICER) (although it works for jobs which are killed for any reason). This queue is a low-priority queue which submits jobs to otherwise unoccupied nodes, and cancels those jobs if a higher priority job requests those nodes. Cancelled jobs from the scavenger queue are automatically restarted when the resources are available. Implementation of the checkpointing system allowed us to make highly efficient use of the scavenger queue by allowing cancelled jobs to be restarted with minimal loss of progress.

Parallel Annealing

GOFISH has an optional run mode that employs a version of parallel annealing to improve the quality of the fit, at the cost of increased computational time. In this mode, multiple swarms are created with different inertial coefficients. At predetermined points during the optimization, the swarms are swapped between layers, changing the inertial coefficient for that swarm.

The GOSIAManager Class

The GOSIAManager class serves as the interface between GOSIA and GOFISH. GOSIA performs solves the coupled differential equations for Coulomb excitation to evaluate the candidate solutions, while GOFISH performs the optimization. The functions in the GOSIAmanager class allow us to parse the GOSIA input and output files to get information, and to create/modify the inputs for evaluating the candidate solutions. The available functions are as follows:

__init__ - The class initialization function takes the config file, detailed in the previous section, as an input. It parses the config file and assigns the user-specified parameters to the proper variables. It then parses the beamINTIinp and beamPOINinp files (and the target versions, if applicable) to get the names of several additional GOSIA input/output files. It also checks if all required arguments are included, and will raise an exception if they are not.

getMatrixElements - This function takes a GOSIA input file as the only argument. It will parse the given file and return a pandas dataframe containing the matrix elements listed in that file, with entries for multipolarity, inital and final states, and the upper and lower bounds assigned to that matrix element.

make_bst - This function creates the file from which GOSIA reads the matrix elements when OP,REST is called. It takes as arguments the name of the file and a list of the matrix element values, in the order they appear in they need to appear in the file. Note that the name specified should match the OP,FILE section of the GOSIA input file.

read_bst - This function reads the file from which GOSIA reads the matrix elements when OP,REST is called. It takes the file name is an input and returns a list of the matrix elements.

getExperimentalObservables - This function populates four arrays with information about the experimental observables by parsing the following files, which must be defined in the config: beamYields, beamINTIinp, and beamPOINinp. Additionally, if simulMin is set to True, it will also parse targetYields, targetINTIinp, and targetPOINinp. The four populated arrays are observables, uncertainties, beamExptMap, and targetExptMap. The observables array contains the values for each experimental observable and literature constraint, and the uncertainties array contains the corresponding uncertainties. The ordering of these arrays will be first the beam observables (in the

order they appear in the GOSIA output file), followed by beam literature constraints, and then target observables and literature constraints if applicable. The beamExptMap and targetExptMap arrays are lists of tuples, with each tuple containing three values: the experiment number (as it appears in GOSIA) and the initial and final state numbers for the associated observable. For literature constraints, the tuple value is (0,0,0). The experiment maps are constructed from the beamYields and targetYields files, which must be formatted in a particular way, described in the section on file preparation. The observables and uncertainties are read from the corrected yields file. The name of the corrected yields file is not a required input to this program. Instead it will be read from the OP,FILE section of the beamINTIinp (or targetINTIinp) file. Literature constrains are parsed from the beamPOINinp and targetPOINinp files. Parsing the constraints requires the inclusion of comments on specific lines in the GOSIA input files, described in the section on file preparation.

getPOINobservables - This function parses a GOSIA OP,POIN output file (which must be passed as an argument) and returns the computed observables for the input matrix elements. The computed observables will be a list in the same format and order as the observables array returned by getExperimentalObservables.

getUpperLimits - This function takes as arguments the beamExptMap, targetExptMap (even if empty), and the observables array, and returns an upper limit for each unobserved transition included in the optimization. It does this by parsing beamPOINinp (and targetPOINinp, if applicable) and reading the upper limit settings specified in the file. Parsing this information requires the inclusion of comments on specific lines in the GOSIA input file(s), described in the section on file preparation.

createSubDirectories - This function creates a sub-directory in the user-defined scratch directory and copies all of the files required for executing the GOSIA **OP,POIN** input file. This is done so that multiple instances of GOSIA can be run simultaneously,

in different directories so that they are not writing to the same output files. This function requires an integer argument, which will be used in the naming of the subdirectory.

removeSubDirectories - This function is used for removing a sub-directory, and allows GOFISH to clean up its temporary files when they are no longer needed. It requires an integer argument which matches the integer used to create the subdirectory in the first place.

runGosia - This function instantiates a subprocess to run GOSIA in the current working directory. The thread will wait for the subprocess to terminate before executing any more lines of code. This function function requires the GOSIA input file to be passed as an input.

runGosiaInDir - This function instantiates a subprocess to run GOSIA in a user-specified directory. As with runGosia, the thread will wait for the subprocess to terminate before executing any more lines of code. The function requires the GOSIA input file and the path to the run directory to be passed as input.

getScalingFactors - This function parses the user-defined scaling factors from the GOSIA **OP,POIN** input file. It does not require any arguments. It will return two lists, the first containing the normalizing experiment number for each experiment (the value of the variable LN from GOSIA's EXPT section) and the second containing the normalization constants for each experiment (the variable YNRM from GOSIA's **OP,YIEL** section). The latter requires comments on specific lines of the GOSIA input file, described in the section on file preparation. This function is only called if simulMin is false. If it is true, scaling factors are instead constrained by the second set of yields.

getAverageAngle - This function parses the average angle for each experiment from the **EXPT** section of the GOSIA **OP,POIN** input file. It does not require any argu-

ments. This information is used to apply corrections to the scaling factors produced by the getScalingFactors function so that we match the calculations done by GOSIA.

getDsig - This function parses a GOSIA OP,POIN output file to get the value of the DSIG parameter for each GOSIA experiment. DSIG is an internal GOSIA parameter that is not user-specified and is used for computing the scaling factors. GOSIA does not print DSIG by default, but the version of GOSIA included in the github repository for GOFISH is modified to do so. This function takes the output file as an argument.