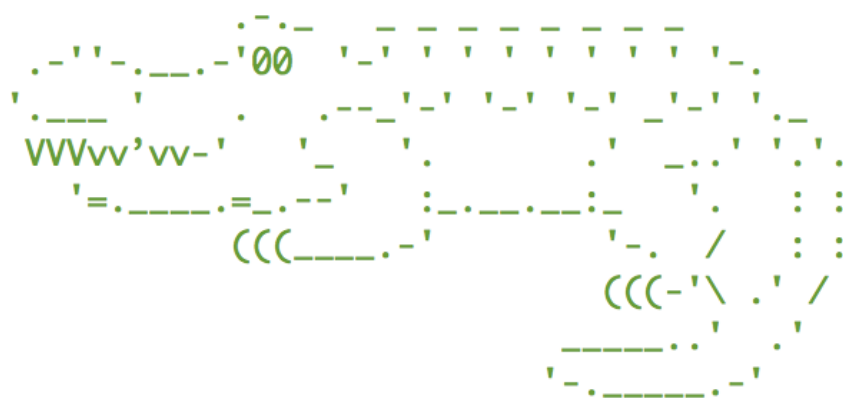


GAtor Genetic Algorithm for Molecular Crystal Structure Prediction



A User's Manual

Marom Research Group

Department of Physics and
Department of Materials Science and Engineering
Carnegie Mellon University, Pittsburgh, PA 15213, USA

April 3, 2018

Contents

1	Basic Installation and Tutorial	1
1.1	Introduction	1
1.2	Installation Requirements for GAtor	1
1.3	Structure of the Code	2
1.4	Running GAtor	2
1.5	Basic Tutorial	2
1.5.1	The ui.conf file	2
1.5.2	Basic ui.conf file settings	2
1.5.3	Filling the initial pool	7
1.5.4	Running the GA	7
1.5.5	Individual and Combined Replica Outputs	7
1.5.6	GAtor time log	8
1.5.7	Structures directory	8
1.5.8	Temp Directories for FHI-aims evaluations	8
1.5.9	Energy Hierarchy	9
1.5.10	Duplicates	9
1.5.11	Finishing the GA	9
1.5.12	Reseting the Calculation Folder	9
2	Full Configuration File Parameters	10
3	Running GAtor in Parallel on Supercomputing Resources	30
3.1	Running GAtor at ALCF	30
3.1.1	Mira	30
3.1.2	Theta	31
3.2	Running GAtor at NERSC	32
3.2.1	Edison	32

Chapter 1

Basic Installation and Tutorial

1.1 Introduction

Welcome to the GAtor genetic algorithm for molecular crystal structure prediction. GAtor uses principles from evolutionary theory such as survival of the fittest, crossover, and mutation that are implemented as operators acting on individual molecules and/or lattice vectors of the fittest crystal structures selected for mating. Energy evaluations and structural relaxations are performed using dispersion-inclusive density functional theory (DFT). For this purpose, GAtor currently interfaces with the all-electron numerical atom-centered orbital DFT code FHI-aims.

1.2 Installation Requirements for GAtor

GAtor is written in Python and interfaces with the all-electron electronic structure theory code FHI-aims. GAtor can be downloaded from <http://software.noamaron.com>. To run GAtor the user will need to install:

- Python 2.7 (<http://www.python.org>).
- NumPy version ≥ 1.9 (<http://www.numpy.org>).
- Pymatgen version $\geq 4.4.0$ (<http://www.pymatgen.org>).
- Sci-kit learn version $\geq 0.17.1$ (<http://scikit-learn.org>).
- A build of FHI-aims (<https://aimsclub.fhi-berlin.mpg.de>), preferably compiled with MPI and scalapack.

The python modules can be installed using, for example, pip. Check that python and the dependent packages can be successfully imported:

```
$ python
>> import numpy
>> import sklearn
>> import pymatgen
```

1.3 Structure of the Code

GAtor contains the main directories `/src` and `/tutorial`. The `/src` folder contains the master script `GAtor_master.py` along with core modules of the GA such as `/selection` and `/crossover`. The `/tutorial` folder provides an example GA run (see Section 1.4).

1.4 Running GAtor

To run GAtor, one runs the master script and inputs a user-defined configuration file `ui.conf` via

```
$ python ../src/GAtor_master.py ui.conf
```

For an explanation of the `ui.conf` file see Section 1.5.1.

1.5 Basic Tutorial

This section takes the user through the example calculation provided in `/tutorial`. This folder contains a `ui.conf` file, a sample initial pool, and sample FHI-aims control files `control.in.SPE.tier_1.dummy` and `control.in.FULL.tier_1.dummy`. The first control file is used just for single point energy evaluations, while the second is used for local optimization¹. The energy cutoffs corresponding to each control file are detailed in Section 1.5.2.

1.5.1 The `ui.conf` file

The `ui.conf` is the only file the user needs to modify in order to use GAtor. It can be named anything as long as it ends with “.conf”. A simple example for the molecule 3-4-cyclobutylfuran can be found in `/gator/tutorial/ui.conf`. The conf file contains the parameters that control all aspects of GAtor including parallelization options, paths to the user-input initial pool, options for interfacing with FHI-aims, and tuning parameters for GA tasks such as mutation probability and duplicate-check tolerances. For an explanation of the simple keywords shown in `/gator/example_calc/ui.conf` see Section 1.5.5. For a full catalog of all the possible keywords that can go into this file, see Chapter 2.

1.5.2 Basic `ui.conf` file settings

This section details the parameters you will see in the basic configuration file `ui.conf`. All parameters are grouped into main sections. This conf file runs GAtor for the molecule 3-4-cyclobutylfuran.

¹To accelerate the DFT calculations for tutorial purposes, limitations are placed on the number self-consistent iterations and max relaxation steps in the control files. These control files should not be used for production purposes, as the calculations will not be fully converged.

[GAtor_master]

The GAtor_master section controls the main procedures of initial pool filling and running the GA.

- **fill_initial_pool = TRUE**
 - Fills the user-defined initial pool into the common pool of structures before running the rest of the GA tasks.
- **run_ga = TRUE**
 - Executes of the main GA procedure

[modules]

The modules section details the names of the individual GA modules used in the sub-folders of /src/. Some of these modules (e.g. selection_module=tournament_selection) may be set to alternative options (e.g. selection_module=roulette_selection). For more information on alternative modules see Section 2.

[initial_pool]

- **user_structures_dir = initial_pool**
 - Path to the pre-prepared initial pool, as generated by *Genarris*. Structures are in a JSON format.
- **stored_energy_name = energy_PBE_TS_tier1**
 - Stored energy name in initial pool json files (if other than "energy"). For production calculations, this should be at the same level of theory as the last control file listed in **control_in_filelist**.

[run_settings]

This section controls general GA run settings.

- **num_molecules = 4**
 - This is the number of molecules in the unit cell to be run (must match number of molecules per unit cell in the initial pool). This must be specified by the user.
- **end_GA_structures_added = 5**
 - Setting which stops GAtor after a certain amount of children (in this case 5) have been added to the common pool. For production calculations this may be set to approximately 250-400 structures. For more options for stopping/converging the GA refer to Section 2.

- `output_all_geometries = TRUE`
 - Prints the FHI-aims-style geometries of parents, children, and mutations to the main output `GAtor.out`. This setting is set to `TRUE` for easy visualization of the structures using Jmol (copy/paste) but may be uncommented for a less verbose output.
- `#skip_energy_evaluations = TRUE`
 - This parameter is uncommented by default, but may be used to skip FHI-aims energy evaluations of the generated structures (giving them an energy of 0 eV). This can be used to verify the structure selection and generation works (e.g. on a laptop) without having to run FHI-aims.

[parallel_settings]

- `parallelization_method = subprocess`
 - The parallelization setting of the GAtor replica(s) being run (not for FHI-aims). Subprocessing uses Python's Subprocess module (to run FHI-aims) and can be used on a laptop, or on a cluster (where the Python subprocess will run on the job scheduler nodes). See Chapter 2 for alternative options.
- `number_of_replicas = 1`
 - Number of GAtor replicas being run by user.
- `processes_per_replica = 1`
 - Number of parallel Python processes used per replica. This sets the number of python processes used for parallel GA tasks such as child generation. This makes the child generation procedure faster but should be set with an awareness of the number of processes available on the given machine being used.
- `aims_processes_per_replica = 64`
 - Number of parallel processes used per replica to run FHI-aims for a given replica. For example, this is set to 64 if one is running 1 replica on 1 node of a cluster with 64 processes (e.g. Theta at ALCF).

[FHI-aims]

- `execute_command = mpirun`
 - Command to run the FHI-aims binary. Since we will be using the scalable version of aims the command is `mpirun`.
- `path_to_aims_executable = aims.mpi.x`

- Path to FHI-aims executable being used for energy evaluations and/or structural relaxations.
- `control_in_directory = control`
 - Name of the directory in the main calculations folder which can contain multiple FHI-aims control.in files (which can be named arbitrarily).
- `control_in_filelist = control.in.SPE.tier_1.dummy control.in.FULL.tier_1.dummy`
 - Name of control file(s) in control directory being used within GAtor. The GA evaluates the control files in order, and the user can set energy cutoffs for each control file (see below). For the tutorial a dummy single-point energy and full relaxation control file are included for demonstrative purposes (i.e. for fast DFT evaluations that are not fully converged). DO NOT use these control files for production calculations.
- `store_energy_names = energy_SPE_tier1 energy_relaxed_tier1`
 - Name of energy names stored for each structure according to the control file(s) in `control_in_filelist`. If these aren't specified only the energy corresponding to the last control file in `control_in_filelist` is stored, and it is stored as "energy".
- `relative_energy_thresholds = 10 10`
 - Relative energy cutoffs (in eV) from the current global minimum structure for each control file specified in `control_in_filelist`. If a structure has a relative energy greater than the minimum energy structure plus this cutoff, it is immediately rejected. For tutorial purposes these are set to relatively large cutoffs. For more energy cutoff options see Section 2.
- `save_failed_calc = TRUE`
 - If uncommented, saves aims calculations if they fail for some reason in `/tmp`
- `save_successful_calc = TRUE`
 - If uncommented, saves full aims calculations data for successful GA structures. Should be commented out if space is an issue.

[selection]

This section controls parameters related to the specific `selection_module` chosen.

- `tournament_size = 3`
 - For tournament selection, this controls the tournament size.

[crossover]

This section controls parameters related to the specific `crossover_module` chosen.

- `crossover_probability = 0.5`
 - This parameter controls the probability of crossover for a child structure. If set to 0.5 the child has a 50% chance of undergoing crossover, and a 50% chance of undergoing mutation. A separate parameter is not needed for mutation.

[mutation]

This section controls parameters related to the specific `mutation_module` chosen.

- `stand_dev_trans = 0.5`
 - Standard deviation (in Angstrom) of the random translation mutations applied.
- `stand_dev_rot = 30`
 - Standard deviation (in degrees) of the random rotation mutations applied.
- `stand_dev_strain = 0.3`
 - Standard deviation of the random strain mutations applied.

[cell_check_settings]

This section controls parameters related to the geometric constraints of generated crystal structures. Structures are rejected if they don't pass these constraints.

- `target_volume = 473`
 - The mean `target_volume` for generated structures
- `volume_upper_ratio = 1.4`
 - The upper ratio of `target_volume` accepted for generated structures.
- `volume_lower_ratio = .6`
 - The lower ratio of `target_volume` accepted for generated structures.

1.5.3 Filling the initial pool

An initial pool of structures, prepared by the user using the *Genarris* molecular crystal generation package is required to run GAtor. For more information on generating this initial pool see the *Genarris* user's manual. The path to this initial pool of structures is set in the `ui.conf` file in `[initial pool]/user_structures_dir`. To start, comment out `run_GA = TRUE` and comment `fill_initial_pool = TRUE`. This will just fill the initial pool without running the GA. Then run the master script

```
$ python ../src/GAtor_master.py ui.conf
```

If the initial pool has properly been filled, one should see a nonempty file in `/tmp/num_IP_structs.dat` that contains the number of nonduplicate initial pool structures.

1.5.4 Running the GA

Once the initial pool has been filled you may run the GA by uncommenting `run_GA = TRUE`. One *could* run the code in-shell (not recommended) by running again

```
python ../src/GAtor_master.py ui.conf &
```

Putting the `&` at the end of the script allows the code to run in the background and frees up your terminal. However, this may take quite a while to finish as FHI-aims calculations are being performed. Therefore, it is highly recommended to submit this command to a cluster. An example `submit_to_cluster.sh` is provided in the tutorial folder. Make sure the number of `[parallelization_settings]/ aims_processes_per_replica` is set in accordance with the number of processes allocated on the cluster.

One GAtor replica is now running! The next step is to look at the different output files being produced.

1.5.5 Individual and Combined Replica Outputs

The output of an individual replica is stored in, e.g.,

```
./tmp/replica_out/fa2f201fe6.out
```

The label is generated randomly, so yours won't be labelled "fa2f201fe6" but some other random hexadecimal. This replica output file records information from the genetic algorithm tasks from each iteration of an individual replica and is reset when either a structure is rejected or successfully accepted. This file includes details from selection, crossover, mutation, comparison, and FHI-aims evaluation for the current structure for the given replica.

The combined output from all successful iterations of all running replicas is stored in.

```
./GAtor.out
```

This combined output file is written to every time any replica starts and finishes an iteration. Feel free to inspect this file as GAtor runs. While this output details various procedures being performed by the GA, it is mostly just for the user's information. Since in the example `ui.conf` it was elected to output all FHI-aims geometries to the replica outputs, you may choose to visualize the geometries from the most recent parents, children, and mutations by copy/pasting their FHI-aims geometries from the replica output file into Jmol. The main data (structures) being produced from the GA run are located in `./structures/C:24_H:24_O:4/0` (See Section 1.5.8)

1.5.6 GAtor time log

A time log that mainly entails information on the execution of FHI-aims energy evaluations for all replicas is stored in,

```
./GAtor.log
```

The user can refer to this file for inquiring when the latest FHI-aims evaluation for their replica has started and stopped as well as if the jobs have failed(hopefully not...).

1.5.7 Structures directory

The database of the entire common pool of the genetic algorithm is located in the the directory, e.g.,

```
./structures/C:24_H:24_O:4/0
```

Each subfolder in this directory corresponds to one structure in the pool, and they are named according to random-indices (if they are an initial pool structure their original name is used). FHI-aims geometries as well as JSON files (which store the structure's geometry and properties) are stored in these files. Feel free to inspect any of these directories.

1.5.8 Temp Directories for FHI-aims evaluations

The currently-running FHI-aims calculation folders (when a new structure has been successfully generated) are located in a directory named after the replica, e.g.

```
./tmp/fa2f201fe6
```

If you change into this directory you will find the `control.in`, `geometry.in`, and `aims.out` files for the currently-running FHI-aims calculation for your replica, as well as a JSON file which includes properties of the currently running structure. The user can inspect `aims.out` if they wish to know exactly where an FHI-aims evaluation is at.

1.5.9 Energy Hierarchy

An energy hierarchy, which ranks structures from the database by their energy is updated in,

```
./tmp/energy_hierarchy_C:24_H:24_O:4.dat
```

If you inspect this file, you will see it includes key information from each structure in the collection including their energy ranking, the size of the pool when they were added (initial pool structures have a value of 0, and GA structures indicate the size of the collection when the structure was added), which replica they came from, their structure index, their energy, their unit cell volume and parameters, and their spacegroup. Additionally, for GA-added structures, information about the mutation procedures performed to generate the structure, as well as the indices of the structure's parents, are included. This file is often the simplest one to look at to see if new structures have been added, and where they fall energy-wise in the collection.

1.5.10 Duplicates

An essential part of any genetic algorithm is the identification of duplicate structures as they are inevitably generated in random crossover processes. The database of structures that are deemed as duplicates (and not included in the common pool) are found in,

```
./structures/S:6_C:16_N:8/duplicates
```

Within the GA, GAtor uses *pymatgen's* `StructureMatcher` class to identify duplicate structures within a user-defined energy window. For more information on changing these duplicate tolerances from their default values, see the user manual.

1.5.11 Finishing the GA

GAtor will stop when the time limit on the computing cluster has expired, or when the convergence criteria is reached. In the tutorial we elected for the GA to stop when 5 structures have been added to the common population (in addition to the initial pool structures). This was set via `end_GA_structures_added = 5`

1.5.12 Resetting the Calculation Folder

To reset the current calculation folder (remove all generated structures and start over) one can just remove the generated `./tmp` and `./structures` directories as well as any GAtor outputs. A script which does this is included as `./reset_calc.sh`. This can be used also if errors happen (due to user mistakes) and one just wants to start fresh.

Chapter 2

Full Configuration File Parameters

The configuration file `ui.conf` (or `[user_defined_label].conf`) is the only file the user has to modify to control all parameters used in GAtor. Listed below are all the possible parameters for `ui.conf`, listed under their respective section headings.

[GAtor_master]

The `GAtor_master` section controls the main procedures of initial pool filling and running the GA.

- `fill_initial_pool =` (optional; Boolean)
 - If present, fills the user-defined initial pool into the common pool of structures before running the GA. The user should omit this keyword if the pool has already been filled and there are just desiring adding another replica to write to the common pool.
- `run_ga =` (optional; Boolean)
 - If present, enables execution of the main GA procedure. See the `parallel_settings` section for details on spawning additional replicas of GAtor.

[modules]

This section details the names of the individual GA modules used in the subfolders of `/src/`. Some of these modules (e.g. `selection_module=tournament_selection`) may be set to alternative options (e.g. `selection_module=roulette_selection`). If the user does not specify any or all of the modules the defaults are used.

- `initial_pool_module = IP_filling`
 - The default initial pool filling module. Currently, this is the only module available. See section `[initial_pool]` for other keywords used with this module.

- `optimization_module = FHI_aims`
 - The default optimization module that sets up, runs, and extracts data from FHI-aims when used for energy evaluations and/or local optimizations. See section `[FHI-aims]` for other keywords used with this module. `tournament_selection`
- `comparison_module = structure_comparison`
 - The default comparison module that uses Pymatgen’s StructureMatcher to compare structures to the common pool before and after local optimization.
- `selection_module = roulette_selection` or `tournament_selection`
 - `roulette_selection` is the default selection module that uses roulette-wheel selection. The other option is `tournament_selection` that selects the winner and runner-up of a tournament size of `[selection]/tournament_size`. See section `[selection]` for other keywords that can be used with this module.
- `clustering_module = cluster` (optional; omit if not used)
 - when the clustering module is present along with `[clustering]/cluster_pool=TRUE` then a cluster-based fitness scheme is performed. See section `[clustering]` for other keywords used within this module.
- `mutation_module = standard_mutation`
 - Currently `standard_mutation` is the default mutation module. See section `[mutation]` for a list of keywords that can be used with this module.
- `crossover_module = standard_crossover` or `symmetric_crossover`
 - `standard_crossover` is the default crossover module. `symmetric_crossover` may also be used. For a list of keywords used in conjunction with these modules see section `[crossover]`.

`[run_settings]`

This section sets main parameters and run settings used across different modules of the GA. Here one can specify, e.g. how many molecules are in the unit cell, how to stop the GA, and the verbosity of the outputs.

- `num_molecules =` (required; integer)
 - Number of molecules per unit cell for the current search (must match number of molecules in initial pool structures).
- `orthogonalize_unit_cell =` (optional Boolean; default `TRUE`)

- If `TRUE` will perform Niggli reduction on all structures (`TRUE` by default, so may be omitted). Else set to `FALSE` to prevent standardization of the crystal lattices (not recommended).
- `end_GA_structures_added =` (optional; integer)
 - A simple way to end the GA by stopping after this many structures have been added by the GA.
- `end_GA_structures_total =` (optional; integer)
 - A simple way to end the GA by stopping after this many structures total structures are in the common pool. This includes the structures added by the GA and the structures in the initial pool.
- `followed_top_structures =` (optional, must be used with `max_iterations_no_change`; integer)
 - Track the top number of structures (as ranked by their energy) to see if they have changed in `max_iterations_no_change`. This is a way of determining convergence.
- `max_iterations_no_change =` (optional, must be used with `followed_top_structures`; integer)
 - If `followed_top_structures` hasn't changed in `max_iterations_no_change`, then stop the GA.
- `verbose=` (optional Boolean; default `TRUE`))
 - If `TRUE`, include for detailed information printed to outputs.
- `output_all_geometries =` (optional; Boolean, set to `TRUE` or omit)
 - Set to `TRUE` to enable replica output of FHI-aims style geometry whenever a new trial structure is generated or altered.
- `failed_generation_attempts =` (optional; default = 100)
 - Number of attempts allowed for the structure generation scheme to fail (e.g., failed cell check) before an error is raised.
- `restart_replicas =` (Boolean; default `TRUE`))
 - If `TRUE` will restart all unfinished DFT calculations in `/tmp`.
- `optimization_style =` (optional; options = ("maximize","minimize"); default = "minimize"))
 - Minimize or maximize the fitness function.

- `property_to_optimize =` (optional; default = "energy")
 - Currently energy is the only available property to optimize.
- `skip_energy_evaluations =` (Boolean; default FALSE)
 - This parameter may be used to skip FHI-aims energy evaluations of the generated structures (giving them an energy of 0 eV). This can be used to verify the structure selection and generation works (e.g. on a laptop) without having to run FHI-aims or the optimization module.

[parallel_settings]

- `parallelization_method =` (optional; default = `serial`)
 - `serial` With this setting only one GA replica which reads and writes to the common pool is spawned. If this setting is used no additional keywords need to be specified in `[parallel_settings]`. If desired, additional simple multiprocessing can be used within the single replica (for parallel python processes such as child creation) by setting `processes_per_replica`. Make sure to not oversubscribe processes of the master node (especially log-in nodes).
 - `subprocess` With this setting the user can spawn several replicas of the GA in the master node (or where GAtor is running) using Python subprocessing. This setting also requires `number_of_replicas` to be set.
 - `mpirun` With this setting the user can spread several replicas of the GA across multiple computing cores or nodes using the `mpirun` command. This setting requires additionally setting at least one of the following: `number_of_replicas`, `processes_per_replica`, or `nodes_per_replica`. If only one of these options is specified, GAtor will automatically calculate the others based on the available resources. If more than one of these options is specified, GAtor will check compatibility of the parameters with the system and proceed. Below are a few common scenarios in a sample job which has been submitted to 20 nodes with each node having 20 processes per node.
 - * The user specifies `number_of_replicas = 10`. GAtor will allocate 2 nodes and 40 processes total for each of the 10 replicas.
 - * The user specifies `number_of_replicas = 40`. GAtor will allocate 10 processes for each of the 40 parallel replicas. This means 2 replicas will be running per node.
 - * The user specifies `number_of_replicas = 3`. GAtor will allocate 7 nodes = 140 processes each for 2 replicas, and 6 nodes = 120 processes to 1 replica.
 - * The user specifies `processes_per_replica = 10`. GAtor will spawn 40 replicas (with 2 replicas assigned to each node) with 10 processes per replica.

- * The user specifies `processes_per_replica = 30`. GAtor will spawn 10 replicas, each assigned 2 nodes, but each replica only being assigned 30 processes each (used e.g. for memory requirements). User has to specify `additional_arguments` in order for the 30 processes to be evenly distributed across the 2 nodes (e.g. `-rr` for round-robin).
 - * The user specifies `processes_per_replica = 6`. GAtor will spawn 60 replicas, assign 3 replicas to each node, and allocate 6 processes to each replica.
 - * The user specifies `nodes_per_replica = 4`. GAtor will spawn 5 replicas, and allocate 4 nodes (80 processes) to each replica.
 - * The user specifies `processes_per_replica = 20` and `nodes_per_replica = 2`. GAtor will spawn 10 replicas, each allocated 2 nodes with 20 processes total. The user has to specify `additional_arguments` (e.g., `-rr` for round-robin) in order for the 20 processes to be evenly distributed across the 2 nodes.
 - * The user specifies `number_of_replicas = 5` and `nodes_per_replica = 2`. GAtor will spawn 5 replicas, each allocated 2 nodes and 40 processes total.
 - * The user specifies `number_of_replicas = 20` and `nodes_per_replica = 1` and `processes_per_replica = 15`. GAtor will spawn 20 replicas, each on 1 node with 15 processes.
- If `ValueError` is raised when using `mpirun` for a job submitted to, e.g., 20 nodes with 20 processes per node, it is possibly caused by scenarios similar to the following:
 - * The user specified `number_of_replicas = 10` and `nodes_per_replica > 2`. GAtor will raise a `ValueError` for oversubscription of nodes.
 - * The user specified `number_of_replicas = 10` and `processes_per_replica > 40`. GAtor will raise a `ValueError` for oversubscription of processes.
 - `srun` With this setting the user can spread several replicas of the GA across multiple computing cores or nodes using the `srun` command. The same parallelization procedure is used as with the setting `mpirun`. See the description for `mpirun` for parameters requirements and how nodes and processes are distributed to each replica.
 - `mira` Special implementation for ALCF's IBM BG/Q cluster Mira. Required additional parameter: `nodes_per_replica`. Additional Python instances of GAtor will be spawned through subprocess on the front-end nodes. The blocks and corners in the back-end nodes are automatically assigned to each replica. Each replica can be assigned more front-end processes by the
 - `processes_per_replica` parameter.
 - `cetus` Special implementation for ALCF's IBM BG/Q testing cluster Cetus. Required additional parameter: `nodes_per_replica`. See the setting `mira` for further details. Different from the `mira` setting in that by default, blocks of 128 nodes are created, instead of 512.

- `python_command` (optional; default: `python`)
 - The command used to call Python. This parameter can be set to call an alternative version of Python.
- `number_of_replicas`
 - Required in "subprocess" parallelization mode; optional in "mpirun" and "srun"; ignored in "mira" and "cetus"
 - Number of parallel replicas running the GA.
- `processes_per_replica` (optional)
 - Number of processes allocated to each replica.
 - In "subprocess", "mira" or "cetus" parallelization modes, defaults to 1.
 - In "mpirun" and "srun" modes, defaults to be calculated according to other specified parameters. (See description above about the `mpirun` mode).
- `processes_per_node` (optional)
 - A further constraint on the size of a multiprocessing pool of workers that each replica can spawn. Useful when replicas control more than 1 node to constrain the amount of workers spawned on the main node. The smaller between `processes_per_replica` and `processes_per_node` determines the size of the `multiprocessing.pool`.
 - Honored only in "mpirun" and "srun" parallelization modes.
 - Defaults to the value obtained through mpirun a Python test code on a node.
- `allocated_nodes` (optional)
 - Nodes allocated for this replica. While additional replicas are spawned, this value is set internally to allocate nodes to each replica.
 - Honored only in "mpirun" and "srun" parallelization mode
 - Defaults to the returned value of the function, `parallel_run.get_all_hosts()`.
- `replica_name` (optional; default: "master"):
 - Name of the currently running process.
 - A random replica name is assigned while internally spawning replicas, or when the main GA processes begin with this parameter still being the default "master" (to avoid conflict of names).
- `im_not_master_replica` (optional; Boolean):

- If present and set to TRUE, suppresses all initialization information printed to time log.

Here are a few parameters specifically set for the implementation on system using the `srun` command. Note that overcommitting memory resources will lead to job unable to run. To successfully run on system with `srun`, make sure to allocate the necessary general resources in the submission file.

- `srun_max_runtime` :
 - Maximum run time in seconds before the master process kills the job
- `srun_gator_memory` (optional; default = 2048):
 - Memory (in MB) devoted to the Gator python processes spawned in a different node.
- `srun_memory_per_core` (optional; default = 1024):
 - Memory per core (in MB) devoted to additional `srun` processes (e.g., for FHI-aims calculations).
- `srun_command_file` (optional; default = `./srun_calls.info`)
 - The path to the file where each replica sends an `srun` call's command to be picked up by the master thread that spawned all the replicas. This is necessary because `srun` does not allow nested calls.
- `srun_submitted_file` (optional; default = `./srun_submitted.info`):
 - The path to the file where the internal job id of `srun` calls that are picked up by master process and executed is recorded
- `srun_completed_file` (optional; default = `./srun_completed.info`)
 - The path where completed commands are sent to notify replicas to pick up results.
- `srun_gres_name` (optional; default = "craynetwork"):
 - Name to the generic resource to that serves as the first field in the argument `-gres` for an `srun` command. Make sure to configure such resources in the original submission file.

Here are a few parameters specifically set for the implementation on IBM's BG/Q system with the `runjob` command:
- `bgq_block_size` (optional):
 - Number of nodes per booted block
 - Defaults to 512 for `mira` mode, 128 for `cetus` mode.

- `runjob_processes_per_node` (optional; default: 16):
 - Number of processes per node. Should be set to the number of cores per node.
- `runjob_block` (optional):
 - For internal distribution of nodes only. The block that is assigned to the replica.
- `runjob_corner` (optional):
 - For internal distribution of nodes only. The corner that is assigned to the replica.
- `runjob_shape` (optional):
 - For internal distribution of nodes only. The shape of the corner that is assigned to the replica.

[FHI-aims]

- `path_to_aims_executable =` (required; `/path/to/aims.x`)
 - Path to FHI-aims executable being used for energy evaluations and/or structural relaxations.
- `execute_command =` (required; `mpirun`, `srun`, `runjob`, or `shell`).
 - Command to run the FHI-aims binary. The shell command should be used when calling a serial version of aims via `/path/to/aims.x`. Note that if `execute_command = shell`, then `additional_arguments` will not be appended to the execute command.
- `additional_arguments=` (optional; not valid if `execute_command = shell`)
 - A Python-evaluable list of strings to append as additional arguments used in the subprocess.Popen call of the FHI-aims binary. For example, set this to `["-rr"]` to enable round-robin spawning method in mpirun. Or set this to `["-envs", "OMP_NUM_THREADS=4"]` to allow the runjob command to alter the environmental variable, OMP_NUM_THREADS. Note that the nodes and processes information are automatically included in the argument list via keywords set in `parallel_settings`.
- `control_in_directory =` (required; `/path/to/control_directory`)
 - Folder name in current directory that holds the control.in files used within Gator.
- `control_in_filelist =` (required; `control.in.1 control.in.2 ...`)
 - File names of control.in files (in `control_in_directory`) used within Gator for successive evaluations of FHI-aims. These can be named arbitrarily in the `control_in_directory`. e.g. One can perform single point calculations with `control.in.1` and full relaxations with `control.in.2`.

- `absolute_thresholds=` (optional; energy1 energy2 ...)
 - List of highest total energies (in eV) allowed for a structure to be deemed as acceptable for each level of `control_in_filelist`. Must match length of `control_in_filelist`. For example, if one does not want to allow into the common pool structures with a single point energy higher than -45,000 eV or a fully-relaxed energy higher than -45,575 eV, then `absolute_thresholds= -45000 -45575`.
- `relative_energy_thresholds=` (optional; rel_energy1 rel_energy2 ...)
 - Energy (in eV) allowed for a structure to be deemed as acceptable for each level of `control_in_filelist`, relative to the current running global minimum. Must match length of `control_in_filelist`. For example, if one does not want to allow into the common pool structures with a single point energy 5 eV higher than minimum energy in the pool or a fully-relaxed energy higher than 3 eV than the minimum energy in the pool, then `relative_thresholds= 5 3`.
- `relative_energy_thresholds=` (optional; rel_energy1 rel_energy2 ...)
 - Energy (in eV) allowed for a structure to be deemed as acceptable for each level of `control_in_filelist`, relative to the current running global minimum. Must match length of `control_in_filelist`. For example, if one does not want to allow into the common pool structures with a single point energy 5 eV higher than minimum energy in the pool or a fully-relaxed energy higher than 3 eV than the minimum energy in the pool, then `relative_thresholds= 5 3`.
- `reject_if_worse_energy =` (optional Boolean list; e.g. `TRUE TRUE`)
 - If present, rejects structures from FHI-evaluations (defined in `control_in_filelist`) if their energy is worse than the worst structure currently in the initial pool.
- `monitor_execution =` (optional Boolean; default = `TRUE`)
 - If present, enables monitoring of the FHI-aims binary call spawned through Python's `subprocess.Popen` module. The monitoring involves: (1) Confirmation of successful job launch, and (2) prevention of job being hung. A job is given 10 attempts to launch before being determined as failed.
- `save_failed_calc` (optional; Boolean)
 - If set to `TRUE`, entire failed FHI-aims calculation folders will be saved to `(./tmp/save_calc_failed)`.
- `save_aims_output` (optional Boolean; default = `FALSE`)
 - If set to `TRUE`, full FHI-aims local optimization outputs will be saved to each structures directory in `/structures/<stoic>/0/<ID>`, `relaxation_data`. As this can take up a lot of storage, also see `save_relaxation_data` as an alternative for saving certain information from the output.

- `save_successful_calc` (optional Boolean; default = `TRUE`)
 - If set to `TRUE`, data extracted from the FHI-aims local optimization outputs will be saved to each structures directory in `/structures/<stoic>/0/,<ID>, relaxation_data`. See `save_relaxation_data` for data that can be parsed and stored. Otherwise necessary information such as a structures energy and geometry are saved from FHI-aims' outputs before the output files are discarded.
- `save_relaxation_data` (optional Boolean; default = all keywords below.)
 - Keywords for data which is stored from FHI-aims local optimization runs. The default keywords are: `cartesian_atomic_coordinates` `Total_energy` `vdW_energy_correction` `Total_atomic_forces` `Hirshfeld_volume` `lattice_vector` `MBD_at_rsSCS_energy` and `Maximum_force_component_after_each_convergence_cycle`. The user can select all or any of these keywords to save data from each FHI-aims SCF relaxation step. This data is stored in each structures directory in `/structures/<stoic>/0/,<ID>, relaxation_data` in JSON format.
- `update_poll_intervals` = (optional if `monitor_execution` = `TRUE`; time; default = 60 seconds per control file)
 - Length of time in seconds to sleep between two checks on the FHI-aims output file. An FHI-aims job must output something within the time period of `update_poll_intervals * update_poll_times`; otherwise, the job is determined to be hung. Must match the length of `control_in_filelist`.
- `update_poll_times` = (optional if `monitor_execution` = `TRUE`; integer; default = 10 per control file)
 - Number of times the FHI-aims output file is polled without new updates before determining that the FHI-aims job has hung. Must match the length of `control_in_filelist`.

[initial_pool]

- `user_structures_dir` = (required; /path/to/user_defined_initial_pool)
 - Path to the user-defined initial pool, as generated by *Genarris*.
- `stored_energy_name` = (required if not stored as "energy")
 - This is the name of the energy stored in the 'properties' of the JSON input files if set as something other than 'energy'. This energy should be at the same level of theory as the last control file listed in `[control]/control_in_filelist`.
- `duplicate_check` = (optional; Boolean; default= `TRUE`)

- If present, will perform a duplicate check on the initial pool of structures by using Pymatgen’s StructureMatcher class. The tolerances used for the duplicate check will be the same as those set in `[post_relaxation_comparison]` or set to the default values.

`[selection]`

This section details the parameters that have to do with the chosen selection module.

- `fitness_function =` (optional; default = `standard_energy`)
 - If the user wishes to use the energy based fitness function, `standard_energy` is set by default. If one wants to use the cluster-based fitness function use `standard_cluster` and the corresponding keywords in the `[cluster]` section.
- `tournament_size` (required if `[module]/selection_module = tournament_selection`)
 - This integer controls the size of the tournament when using tournament selection. Make sure this tournament size is less than the initial pool. For an initial pool with 50 structures typically this is set to 10-20.
- `select_in_cluster` (Boolean; optional if `[module]/selection_module = tournament_selection`; default = `FALSE`)
 - If `TRUE`, will force all selected structure to have the same cluster label as winner of tournament. First the runner up is checked if it has the same cluster label as the winner. If not, another structure with the same group outside of the tournament is selected. If the cluster size of the winner is 1 then the runner up will be selected as the second parent no matter what its cluster label is.
- `fitness_reversal_probability =` (optional; default = 0.0)
 - The user may set this parameter to be between 0.0 and 1.0 to allow a probability of the fitness function being reversed when selecting parents. This may create better diversity in the pools to allow an occasional unfit structure to be selected.
- `percent_best_structs_to_select` (optional; default = 100)
 - The user may set this parameter if they wish to bias selection to only a certain percentage of top fitness structures.

`[pre_relaxation_comparison]`

This section sets the tolerances of Pymatgen’s StructureMatcher class for pre relaxation comparison of a generated structure. For more detailed info on these parameters see the Pymatgen documentation.

- `ltol` (optional; default = 0.2)

- Lattice parameter length tolerance to deem structure as duplicate.
- `stol` (optional; default = 0.4)
 - Atomic site tolerance (RMS) to deem structure as duplicate.
- `angle_tol` (optional; default = 3)
 - Lattice angle tolerance (in degrees) to deem structure as duplicate.
- `scale_vol` (optional Boolean; default = `FALSE`)
 - If `TRUE` will scale structures to equivalent volumes before comparison.

`[post_relaxation_comparison]`

This section sets the tolerances of Pymatgen's StructureMatcher class for post relaxation comparison of a generated structure. For more detailed info on these parameters see the Pymatgen documentation.

- `ltol` (optional; default = 0.2)
 - Lattice parameter length tolerance to deem structure as duplicate.
- `stol` (optional; default = 0.3)
 - Atomic site tolerance (RMS) to deem structure as duplicate.
- `angle_tol` (optional; default = 2.5)
 - Lattice angle tolerance (in degrees) to deem structure as duplicate.
- `scale_vol` (optional Boolean; default = `FALSE`)
 - If `TRUE` will scale structures to equivalent volumes before comparison.

`[cell_check_settings]`

- `full_atomic_distance_check` (optional; default= 0.211672 Angstrom)
 - Enforces a minimum distance between all pairs of atoms in the system. The default value 0.211672 Å is the equivalent of 0.4 bohr, which is the minimum distance enforced by FHI-aims.
- `interatomic_distance_check` = (optional; default= 1 Angstrom)
 - If present, enforces a minimum distance for atom pairs from different molecules. This value should usually be set larger than `full_atomic_distance_check` to enforce a larger distance between atoms from different molecules.

- `COM_distance_check` = (optional; float)
 - If present, enables the COM distance check, which enforces minimum distance between the center of mass of different molecules.
- `specific_radius_proportion` = (optional; default=0.65)
 - A closeness check for potential structures where each atom is assigned a specific radius (by default, their van der Waals radii). In this check, two atoms from different molecules need to be at least a certain proportion (specified by this parameter, which is often shortened as s_r) of the sum of their specific radii apart. E.g. the van der Waals radius of carbon is 1.70 Å, nitrogen's is 1.55 Å; thus if $s_r=0.75$, then any pair of intermolecular C-N contact must be at least $(1.70+1.55)*0.75=2.44$ Å apart.
- `target_volume` = (optional; float)
 - If present, enables volume checks on generated structures. Enforces the volume of a newly generated structure to be within $\text{target_volume}*\text{volume_lower_ratio}$ - $\text{target_volume}*\text{volume_upper_ratio}$.
- `volume_upper_ratio` = (optional; float)
 - The upper ratio that defines the lower bound of the volume of a newly generated structure when times by the `target_volume`.
- `volume_lower_ratio` (optional; float)
 - The lower ratio that defines the lower bound of the volume of a newly generated structure when times by the `target_volume`.
- `lattice_vector_check` (optional Boolean; default = `FALSE`)
 - If present will conduct checks on the generated structures lattice parameters as detailed in the next four keywords.
- `lattice_vector_lower_ratio` (optional; float)
 - If present along with `lattice_vector_check=TRUE` will only allow structures with lattice parameters greater than $\text{lattice_vector_lower_ratio}*V^{1/3}$ where V is the structures unit cell volume.
- `lattice_vector_upper_ratio` (optional; float)
 - If present along with `lattice_vector_check=TRUE` will only allow structures with lattice parameters less than $\text{lattice_vector_upper_ratio}*V^{1/3}$ where V is the structures unit cell volume.
- `lattice_vector_lower_bound` (optional; float)

- If present along with `lattice_vector_check=TRUE` will reject structures with a given lattice parameters smaller than this allowed value (in Angstrom).
- `lattice_vector_upper_bound` (optional; float)
 - If present along with `lattice_vector_check=TRUE` will reject structures with a given lattice parameters greater than this allowed value (in Angstrom).

[crossover]

This section details the various parameters associated with crossover.

- `crossover_probability` (optional; default = 0.75)
 - This parameter controls the probability of crossover (versus mutation) for a child structure. If set to 0.75 the child has a 75% chance of undergoing crossover, and a 25% chance of undergoing mutation. A separate parameter is not needed for mutation. If one wishes to fo 100% mutation then set this parameter to 0.0.

The following set of keywords have to with symmetric crossover and are relevant when `[modules]/crossover_module=symmetric_crossover` and one wants to set them to other values than their default. The term here "seed molecules," means the symmetrically independent molecules (asymmetric unit) within a structure. The symmetric crossover module takes the 1st selected structure as the reference parent and conducts crossover that blends/swaps certain features of the 1st structure with/by that of the 2nd.

- `swap_sym_prob` (optional; default = 0.50)
 - The probability of the symmetry operation of the 2nd structure to be applied to the 1st. In this case, the seed molecules of the 1st structure become those closest, in terms of absolute COM coordinates, to the seed molecules of the 2nd structure.
- `swap_sym_tol` (optional; default = 0.01)
 - Tolerance for determining whether the 2nd structure's symmetry operations are compatible with the 1st structure's lattice vectors.
- `blend_lat_prob` (optional; default = 0.50)
 - The probability for the lattice vectors to be blended during crossover. If without blending, the vectors will be taken straight from the 1st selected structure.
- `blend_lat_tol` (optional; default = 0.01)
 - Tolerance for determining whether the blended lattices are compatible with the symmetry operations.
- `blend_lat_cent` (optional; default = 0.50)

- The center of the Gaussian sampling for the blending parameter, b . Let L_1 be the lattice matrix of the first structure, L_2 be that for the second. Then the blended lattice matrix will be $b \cdot L_2 + (1 - b) \cdot L_1$. Therefore, $b = 0$ takes the unchanged lattice of first structure. $b = 1$ takes the unchanged lattice of second structure.
- `blend_lat_std` (optional; default = 0.25)
 - The standard deviation for the Gaussian sampling of the blending parameter.
- `blend_lat_ext` (optional; Boolean)
 - If set to TRUE, then the blending parameter can be smaller than 0 or greater than 1.
- `blend_mol_COM_prob` (optional; default = 0.50)
 - The probability for the COM of the molecules to be blended during a crossover. During the blending process, each "seed molecule" in the 1st structure will be paired up with a closest neighbor in the 2nd structure, in terms of absolute COM coordinates. If without blending, the absolute COM coordinates will be taken from the 1st selected structure.
- `blend_mol_COM_cent` (optional; default = 0.50)
 - The center of the Gaussian sampling for the blending parameter, b . Let c_1 be the COM of the seed molecule. Let c_2 be the COM of the paired molecule. Then the COM of the seed molecule will be moved to $b \cdot c_2 + (1 - b) \cdot c_1$. Thus, $b = 0$ takes the unchanged COM positions of the seed molecule in the first structure. $b = 1$ takes that of the second structure. If there are multiple seed molecules, b is generated separately for each blending.
- `blend_mol_COM_std` (optional; default = 0.25)
 - The standard deviation for the Gaussian sampling of the blending parameter.
- `blend_mol_COM_ext` (optional; Boolean)
 - If set to TRUE, then the blending parameter can be smaller than 0 or greater than 1.
- `swap_mol_geo_prob` (optional; default = 0.50)
 - The probability for the molecule geometry of the 2nd structure to be swapped into the 1st. The final orientation will be selected from 20 random orientations that have the least coordinate residual from the original geometry in the 1st structure.
- `swap_mol_geo_tol` (optional; default = 3.0)

- The tolerance on coordinate residual in determining whether two molecule conformations are the same. If yes, then the geometry will not be swapped. If all pairs of molecules are the same, then this operation will be ruled invalid.
- `swap_mol_geo_orient_attempts` (optional; default = 100)
 - Number of attempts to randomly orient the swapped geometry. The final orientation is selected to be the orientation that has the least coordinate difference with the original molecule.
- `blend_mol_orien_prob` (optional; default = 0.50)
 - The probability for the orientation of the molecules to be blended during a crossover. During the blending process, each "seed molecule" in the 1st structure will be paired up with a closest neighbor in the 2nd structure, in terms of absolute COM coordinates.
 - If the paired up molecule has different geometry than the seed molecule (see parameter `blend_mol_orien_tol`), then blind blending will be pursued. a number of random rotations (`blend_mol_orien_orient_attempts`) will be applied and the new orientation with the minimum average coordinate difference from the two original molecules will be selected. The average is weighted by the blending parameter b (the coordinate difference from the paired molecule gets weighted as b , while that from the seed molecule gets $1 - b$).
 - The blind blending has a probability specified by `blend_mol_orien_ref_prob` to allow exploration of reflection after applying random rotations. If the exploration is pursued, half of the random rotations will be followed by a mirror reflection across z axis.
 - If the paired up molecule has the same geometry as the seed molecule, then the blending will be based on the calculated mapping information from one to the other. The mapping information gives whether or not a mirror reflection is involved, and a rotation in terms of an axis and an angle in degrees. If the mapping does not involve a mirror reflection, then a portion (b) of the rotation will be applied to the seed molecule as the final rotation.
 - If the mapping involves a mirror reflection, then a mirror reflection is applied with a probability given by `blend_mol_orien_ref_prob`. If the mirror reflection is not applied, then blind blending will be pursued. If it is applied, first the orientation of the reflected molecule that has the smallest coordinate differences from the original will be found (with `blend_mol_orien_orient_attempt` random rotation attempts). Then the mapping information will be recalculated. A portion (b) of the rotation will be applied to the reflected and readjusted molecule geometry as the final orientation. Note that it is likely for the mapping calculation to yield a larger tolerance and thus consider the molecules to be different after reflection is applied. In that case, blind blending will be pursued.
- `blend_mol_orien_cent` (optional; default = 0.50)

- The center of the Gaussian sampling for the blending parameter, b . See description above for parameter `blend_mol_orien_prob` for how b is used.
- `blend_mol_orien_std` (optional; default = 0.25)
 - The standard deviation for the Gaussian sampling of the blending parameter, b .
- `blend_mol_orien_ext` (optional; Boolean)
 - If set to TRUE, then the blending parameter b can be smaller than 0 or greater than 1.
- `blend_mol_orien_tol` (optional; default = 3.0)
 - The coordinate difference tolerance in determining whether the seed molecule has the same geometry as the paired molecule. See description above for parameter `blend_mol_orien_prob` for how this affects the procedure.
- `blend_mol_orien_ref_prob` (optional; default = 0.5)
 - The probability for mirror reflection to be allowed in the final orientation. See description above for parameter `blend_mol_orien_prob` for the usage of this parameter.
- `blend_mol_orien_orient_attempts` (optional; default = 100)
 - The number of attempts to randomly orient a molecule. See description above for parameter `blend_mol_orien_prob` for the usage of this parameter.
- `allow_no_crossover` (optional; Boolean)
 - If set to TRUE, then a crossover attempt that did not invoke any of the above listed operations will be allowed. Otherwise, a `while` loop will be used until 1 attempt uses any of the operations above.

[mutation]

- `double_mutate_prob` = (optional; default=0.05)
 - A user may set this parameter to allow double mutations on crossover structures. If set, the probability of a structure undergoing double mutation is `double_mutate_prob*(1- crossover_probability)`
- `stand_dev_trans` = (optional; default = 3.0 Å)
 - Sets the standard deviation of the random translation mutations to the COM of the molecules in the cell. The translations are randomly picked from a gaussian distribution of this width.
- `stand_dev_rot` = (optional; default = 30 degrees)

- Sets the standard deviation of random rotation mutations to the COM of the molecules in the cell (euler angles).
- **stand_dev_strain** = (optional; default = 0.3)
 - This parameter sets the standard deviation of mutations which involve strain (using a generic strain tensor) on the lattice of the child structure. It's a proportional parameter so, e.g., (default *stand_dev_strain* = 0.3 so for a strain along only the x component of a lattice e.g. $Ax_{strain} = Ax + (0.3 * Ax)$).
- **stand_dev_cell_angle** = (optional; default = 20 degrees)
 - This parameter controls the standard deviation of how much a single lattice parameter angle may change when applying a type of strain mutation where only a lattice parameter angle is changed (and the COM of the molecules are moved accordingly).
- **specific_mutations** = (optional; default = All parameters below)
 - If the user only wants to implement *specific* mutations for a given GA they may list them here separated by spaces. The available options correspond to distinct mutation procedures.
 - The keywords corresponding to distinct mutation operations are listed below.
 - Rand_strain** Apply random strains to the unit cell
 - Sym_strain** Apply random strains to the unit cell symmetrically
 - Vol_strain** Apply random strains to the unit cell which preserve the parent structures unit cell volume.
 - Angle_strain** Apply random strains to the unit cell that only change the a single lattice parameter angle (See **stand_dev_cell_angle**).
 - Rand_trans** Apply random translations to the COM of molecules in the unit cell in Cartesian space.
 - Frame_trans** Apply random translations to the COM of molecules in the unit cell in their own inertial reference frame.
 - Rand_rot** Apply random rotations molecules in the unit cell in Cartesian space.
 - Frame_rot** Apply random rotations molecules in the unit cell in their own inertial reference frame.
 - Permute_mol** Randomly swap the COM of randomly selected molecules in the unit cell.
 - Permute_ref** Randomly swap the COM of molecules of randomly selected in the unit cell and then apply a reflection through each molecules COM about the x y or z directions.
 - Permute_rot** Randomly swap the COM of molecules of randomly selected in the unit cell and then apply a random rotation.

[clustering]

To use a cluster-based fitness function, make sure `[selection]/fitness_function=standard_cluster` is set. To further specify which clustering algorithm and feature vector to use, specify the appropriate keywords in this section.

- `clustering_algorithm` = (optional)
 - The clustering algorithm to be used with `feature_vector`. Currently implemented options include `AffinityPropagation` (recommended) and `Kmeans`.
- `feature_vector` = (optional)
 - The key for the feature vector which is being used for clustering. Currently implemented options include `RDF_vector`, `RCD_vector`, and `Lat_vol_vector`.
- `interatomic_pairs` = (required when `feature_vector = RDF_vector`)
 - Pairs of element types used to compute interatomic distances for the RDF vector. The elements are separated by spaces. So e.g. `interatomic_pairs = C N S S` would compute and bin the interatomic distances for C-N and S-S contacts.
- `interatomic_distance_range` = (required when `feature_vector = RDF_vector`; default= `1 8` corresponding to 1-8 Angstrom)
 - The interatomic distance range sampled when computing the RDF vector of interatomic contacts. Two integers must be present separated by a space.
- `interatomic_distance_increment` = (required when `feature_vector = RDF_vector`; default 1 Angstrom)
 - The interatomic distance interval within the `interatomic_distance_range` used for creating the RDF vector. If set to 1, then `interatomic_distance_range` will be binned in intervals of 1 Angstrom for the construction of the RDF vector.
- `rcd_axes_indices` = (required when `feature_vector = RCD_vector`)
 - The indices of molecules used for the construction of a molecules reference frame/basis vectors for the RCD vector. Only 4 indices have to be provided. For example, if `rcd_axes_indices = 6 9 7 10` then one basis vector will be constructed which points from atom 6 to 9, another will be constructed which points from atom 7 to 10, and the third will be constructed using gram-schmidt (all vectors are orthogonalized by gram-schmidt).
- `rcd_close_picks` = (integer; required when `feature_vector = RCD_vector`)
 - The number of molecules surrounding the reference molecule in the RCD calculation. Default = 8 molecules for 4 molecule per cell crystals.

- `num_clusters =` (integer; required when `clustering_algorithm=Kmeans`)
 - The number of clusters for the Kmeans algorithm. AffinityPropagation doesn't require this parameter to be set, as it find the clusters in the data inherently.

Chapter 3

Running GAtor in Parallel on Supercomputing Resources

Because GAtor’s parallelization is different for different computer architectures and job schedulers, this chapter details how GAtor can be run on different machines at the Argonne Leadership and Computing Facility (ALCF) and the National Energy Research Scientific Computing Center (NERSC).

3.1 Running GAtor at ALCF

3.1.1 Mira

Below is an eample submission script, `Submit_Mira.sh`, which runs GAtor using 512 nodes, a maximum wall-time of 60 minutes, and charging the job to `MyProject`:

```
#!/bin/bash
#COBALT -t 60
#COBALT -n 512
#COBALT -A MIO_HP_2
#COBALT --disable_preboot

python ../src/GAtor_master.py ui.conf
```

The `--disable_preboot` flag must be added to the qsub submission script. The following command submits the script `Submit_Mira.sh` to Mira’s Cobalt job scheduler:

```
qsub Submit_Mira.sh
```

Below is an example of the `[parallel settings]` and `[FHI-aims]` module settings in that are required for runing GAtor on Mira:

```
[parallel_settings]
parallelization_method = mira
```



```

nodes_per_replica = 128
processes_per_replica = 4
runjob_processes_per_node = 16
python_command = python

[FHI-aims]
execute_command = runjob

```

The parameter `parallelization_method = mira` requires the additional parameter `nodes_per_replica`. The number of Gator replicas is then automatically calculated by dividing the number of nodes specified in the submission script by the number `nodes_per_replica`. For this example, Gator would spawn 4 instances. Python replicas of Gator will be spawned through subprocess on the front-end nodes. The blocks and corners in the back-end nodes are automatically assigned to each replica. Each replica can be assigned more front-end processes by the `processes_per_replica` parameter.

The `runjob` command tells Mira to execute the FHI-aims calculation on the back-end nodes. `runjob_processes_per_node` is set to 16 because each Mira nodes has 16 processor leaving 1 GB of memory for each process. However, if memory becomes an issue, the number of processes per node can be decreased.

Note: Access to Mira comes with access to Cetus. The primary role of Cetus is to run small jobs and debug problems that occurred on Mira. Conveniently, Cetus shares the same software environment and file systems as Mira. To run Gator on Cetus, change `parallelization_method = cetus`. Cetus has a maximum wall time of 60 minutes and small jobs (128 nodes) tend to get through almost instantly. You can check the availability of Cetus (and all ALCF resources) at <https://status.alcf.anl.gov/cetus/activity>.

3.1.2 Theta

Below is an example submission script, `Submit_Theta.sh`, to submit Gator to Theta's default queue using 8 nodes, a maximum wall-time of 30 minutes, and charging the project `MyProject`:

```

#!/bin/bash
#COBALT -n 8
#COBALT -t 30
#COBALT -q default
#COBALT -A MyProject

python ../src/Gator_master.py ui.conf

```

The following command submits the script `Submit_Theta.sh` to Theta's job scheduler Cobalt:

```
qsub Submit_Theta.sh
```

The `default` submission queue is for for the entire Theta system. The default queue

has a minimum job time of thirty (00:30:00) minutes and a minimum allocation of 8 nodes. There are two 16-node debugging queues on Theta with maximum wall-clock times of 1:00:00 (1 hour); `debug-cache-quad` and `debug-flat-quad`.

When a batch script is submitted on Theta, it is executed on the Machine Oriented Mini-server (MOM) nodes. Once on the MOM node, the `aprun` command is used to run executables on the compute nodes. The following is an example of the `[parallel settings]` and `[FHI-aims]` module settings in the Gator `ui.conf` file for running on Theta:

```
[parallel_settings]
parallelization_method = subprocess
number_of_replicas = 8
processes_per_replica = 1
aims_processes_per_replica = 64

[FHI-aims]
execute_command = aprun
```

The `subprocess` parallelization setting will spawn `number_of_replicas = 8` (python) Gator instances on the MOM nodes. Simple multiprocessing can be used within the single replicas (for parallel python processes such as child creation) by setting the value of `processes_per_replica`. **Important:** Since the MOM nodes perform many different tasks for Theta, do not set `number_of_replicas * processes_per_replica` to be greater than 32 or you risk crashing these nodes. The Theta compute nodes each have 64 cores so the number of parallel processes used to run FHI-aims per Gator replica is set to 64 processes using the `aims_processes_per_replica` setting. The `execute_command` of FHI-aims for Theta is `aprun` so that energy evaluations and structure relaxations will be performed on the compute nodes.

3.2 Running Gator at NERSC

3.2.1 Edison

Below is an example submission script, `Submit_Edison.sh`, to submit Gator to Edison's debug queue using 4 nodes, a maximum wall-time of 30 minutes:

```
#!/bin/bash
#SBATCH --qos=debug
#SBATCH --time=00:30:00
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=24
#SBATCH --gres=craynetwork:2

module load python
export OMP_NUM_THREADS=1
```

```
python ../src/GAator_master.py ui.conf
```

The following command submits the script `Submit_Edison.sh` to Edison's SLURM job scheduler (to the debug queue in the example script):

```
sbatch Submit_Edison.sh
```

On Edison, the GA replica python processes run in the backend nodes. The `srun` command is used both to spawn the python processes and to launch FHI-aims on the backend.

The following is an example of the `[parallel settings]` and `[FHI-aims]` module settings in the Gator `ui.conf` file for running on Edison:

```
[parallel_settings]
parallelization_method = srun
number_of_replicas = 2
nodes_per_replica = 2
processes_per_replica = 4
aims_processes_per_replica = 40

[FHI-aims]
execute_command = srun
additional_arguments = ["-O"]
```

The `srun` parallelization setting will spawn `number_of_replicas = 2` Gator instances on the backend nodes. Additional multiprocessing can be used within the single replicas (for parallel python processes such as child creation) by setting the value of `processes_per_replica`. In the example script, 2 GA replicas run on 2 nodes each (for a total of 4 nodes allocated for the job). Each replica has 4 python processes for child generation and 40 aims processes for energy evaluations and local relaxations.