# Fritz Haber Institute *ab initio* molecular simulations: FHI-aims

All-Electron Electronic Structure Theory
with Numeric Atom-Centered Basis Functions

A Users' Guide

FHI-aims team
Fritz-Haber-Institut der Max-Planck-Gesellschaft, Berlin
and many contributors around the world.
July 21, 2015

# Contents

# Chapter 1

# Parallel Genetic Algorithm Search

Comments and suggestions to:
newhouse@fhi-berlin.mpg.de
huynh@fhi-berlin.mpg.de
ghiringhelli@fhi-aims.berlin.mpg.de
levchenko@fhi-berlin.mpg.de.

## 1.1 Introduction

!!! complete this section

## 1.2 design of the program

!!! complete this section

modularity

digression from standard genetic algorithms

Almost all of this is contained in poster.

## 1.3 Input

Each execution of the Genetic Algorithm takes place within a working directory. This directory requires the existence of a few files and directories. This section specifies the required inputs.

### 1.3.1 Arguments

To initialize the program, the user may run the master.py script from the working directory. The program can be run with zero, one or two arguments. The full input of the program is as follows:
`/path/to/program/src/core/master.py <user_input> <stoichiometry>` If the second argument is ommitted, the stoichiometry definition is assumed to reside in the user input file. If both arguments are ommitted, the user input filename is assumed to be ui.conf.

There are also other arguments that may be passed to master.py:
clean: removes the structure data, temporary data, and output files, resetting the state of the working directory.
data_tools: runs the data_tools.py module that may be used for visualization and analysis of data (may be executed during GA run).
kill: upon a replica's next iteration, that replica will terminate cleanly. !!! format

### 1.3.2 Given Files

There are a number of files required as inputs to the genetic algorithm which are to be located in the working directory at runtime.

**user input file:** default: ui.conf This file contains all configuration information necessary for a genetic algorithm run. !!! complete

**control.in directory:** !!! complete

**user structures directory:** !!! (Complete this section upon regaining access to fhi filesystem) the control.in files used for each cascade level

## 1.4 Important Data Strucures

The Structure and StructureCollection objects are arguably the most important objects in the GA. It is worthwhile becoming very familiar with them in order to understand the behaviour of the rest of the program.

**Structure** The Structure class represents a specific molecule's geometry, composition, and calculated properties. The structure of the molecule is stored in the geometry field as a numpy array. The specified dtype of the numpy array allows for access of array elements (and columns) by name. Currently the geometry field maintains the following information for each atom: x, y, z (coordinates), element (by symbol), spin, charge, fixed (used for substrates). The class contains methods to construct the geometry from various sources (text files, aims outputs, etc.). To store various properties, a Structure object also contains a dictionary (hash table) named properties in which a key-value pairs are stored for each property. This scheme was employed in order to allow flexibility in storing configuration-specific properties.

**StructureCollection** The StructureCollection class was developed in order to clearly separate groups of Structures based on a) the level of simulation precision when using a cascading GA and b) the stoichiometry of the structures when allowing for stoichiometric crossover. It is also the only class that acts as an interface with the structures stored on the filesystem. The StructureCollection class can read and write files containing all information contained in a Structure object to and from a shared location (filesystem or database (needs to be implemented)). Each structure collection has a unique key composed of its a) its assigned stoichiometry and b) its assigned input reference number (corresponds to the level of cascade).

## 1.5 Usage

This program is separated into three major levels of usability:

#### keyword level

This level is intended for the slight modification of a use case which is already fully implemented. For example, one may wish to change the probability of the mutation of a structure, the acceptance tolerance of newly-made structures, or the precision of relaxation.

#### module level

This level is intended for more advanced usage. Almost every operation of the genetic algorithm is carried out by interchangeable modules. Initial pool filling, random structure generation, crossover, mutation, relaxation (or energy calculation), structure selection. These modules only require that a function named main() exists which accepts specific arguments and returns specific values. Aside from this one requirement, the specific implementation of these modules is entirely up to the user. This case is intended for the range of users who simply would like to introduce a new style of mutation to the users who intend to change the entire type of material being studied.

#### core level

This level is intended for the most advanced usage. A user who changes this implementation should have a full understanding of the program. any changes may break compatibility with existing modules. The core modules include those in the core, and structures packages. These modules are the backbone of the genetic algorithm and handle the calling of the required pieces. They also determine the properties of the "Structure" and "StructureCollection" objects and how the data are shared between replicas.

## 1.6 Keywords

**[section]**

**keyword = default**
(argument type)
explanation

### 1.6.1 core keywords

**[modules]**

The following keywords name the user-implemented modules that will be used in the genetic algorithm. Do not include the '.py' extension.

**initial_pool_module = fill_simple_norelax**

**relaxation_module = FHI_aims_relaxation**

**initial_pool_relaxation_module = reax_relaxation**

**random_gen_module = random_structure_1**

**comparison_module = structure_comparison_1**

**initial_pool_comparison_module = initial_pool_comparison_1**

**selection_module = structure_selection_1**

**mutation_module = mutation_1**

**crossover_module = crossover_1**

**[control]**

The following keywords specify the location of the control files needed for relaxation and optimization (for FHI-aims, LAMMPS, etc).

**control_in_directory = control**
(Any string, must match a directory name in the working directory.)
This is the location containing the files which define the various levels of cascade.

**initial_pool = initial_pool.in**
(Any string, must match a file name in the control_in_directory.)
If using a method of relaxation to pre-relax the initial pool, this is the control.in file which will be used. It corresponds to the cascade level -1.

**control_in_filelist = control.in**
(Any strings separated by whitespace, must match a file name in the control_in_directory.)
This keyword defines the list of control.in files used in the cascade portion of the genetic algorithm. To increase the levels of cascade, simply increase the specified control.in files accordingly. e.g. "control_in_filelist = PBElight.in PBE0tight.in HSE06.in"

**[run_settings]**

The following keywords specify general settings of the GA run.

**number_of_structures = 100**
(Any positive integer.)
The GA run will terminate after this number of structures is added to the highest cascade level.

**number_of_replicas = 1**
(Any positive integer.)
When running the genetic algorithm on a cluster, this specifies how many times master.py's main function will submit to the queue.

**parallel_on_core = None**
(Any positive integer less than or equal to the number of processors available on a single core or None.)
When relaxation is done using a single processor (e.g. force-field calculations) this option allows for multiple replicas to take place simultaneously on a single node. "None" will execute a single replica on a single core.

**recover_from_crashes = False**
(Boolean.)
Set to True to attempt to recover from unexpected exceptions (e.g. caused by relaxation algorithms). Leave False in general and especially for debugging. Replica will restart as if beginning for the first time without raising exception.

**verbose = False** (Boolean.)
Used to set the verbosity level of output. Leave false to return only vital information

### 1.6.2 pre-made module keywords

These keywords are not vital, but correspond to the default modules used for clusters in the genetic algorithm.

### [initial_pool]

The following keywords specify the behaviour of the filling of the initial pool

**num_initial_pool_structures = 6**
(Any non-zero integer.)
Defines the number of randomly generated initial structures are created

**user_structures_dir = user_structures**
(Any string, must match a directory name in the working directory.)
Specifies the location of user-defined initial pool structures. An attempt will be made to add every file in this directory to the initial pool

**number_of_processors = 12**
(Any positive integer less than or equal to the number of processors available on a single core.)
In parallel-process initial pool filling, this number defines the number of processes running simultaneously in the initial pool stage

### [random_gen]

The following keywords specify how structures are randomly generated when filling the initial pool.

**rand_struct_dimension = 3**
(1, 2 or 3)
specifies the space in which the random structures are generated

**minimum_bond_length = 0.5**
(Any positive real number)
When generating structures, this value is used to determine if any of the resulting bonds are too short. measured in angstroms. to be replaced by a distance ratio comparison like that in periodic structures.

**model_structure = None**
(String or None. Must match a filename in the working directory.) This specifies the model that is referenced when seeking specifications for a randomly generated structure

**min_distance_ratio = 0.1**
(positive real number)
This parameter and the elements' radii are used when comparing closeness internally. currently used only in periodic structure generation

### [selection]

The following keywords specify how fitness is calculated and how structures to cross are selected based on that fitness calculation

**fitness_function = standard**
(standard or exponential)
Will calculate the fitness in the standard format or with an exponential weighting

**alpha = 1**
(Any real number.)
A parameter used in exponential weighting

**fitness_reversal_probability = 0.1**
(Any number from from 0.0 to 1.0)
Determines the probability that the fitness will be reversed and unfit structures favored for one selection (introduces diversity)

**stoic_stdev = 1**
(Any positive real number.)
the standard deviation used when selecting across stoichiometries. the larger the number, the more likely the possibility of selecting a different stoichiometry.

**[comparison]**

The following keywords specify how a new structure is compared to the existing structures and whether it is acceptable or not

**always_pass_initial_pool = False**
(Boolean)
Allows for bypassing initial pool comparison in order to accept every generated structure regardless of similarity.

**dist_array_tolerance = 0.5**
(Any positive real number)
The threshold that must be reached for two structures to be deemed different when comparing distance arrays. larger -> stricter. changes with number of atoms. default is a very low threshold.

**energy_comparison = 1.0**
(Any number between 0.0 an 1.0)
A newly relaxed structure will be automatically discarded if the energy is greater than some proportion of the existing structures energies. This number defines that ratio. 1.0: must be at least lower than the highest energy. 0.5: must be at least lower than half the energies. 0.0 must be the lowest energy in the collection.

**energy_window = None**
(Positive real number or None)
When filtering structures to compare at a fine level, this defines the tolerance of filtration based on energy closeness.

**histogram_tolerance = 0.2**
(Any positive real number)

When filtering structures to compare at a fine level, this defines the tolerance of the histogram filter. TODO: explain what happens with higher and lower numbers

**n_bins = 10**
(Positive integer)
number of bins used in histogram comparison

**bin_size = 1.0**
(Positive real number)
size of the histogram bin. measured in angstroms.

**[crossover]**

The following keywords specify how structures are crossed to create a new structure

**crossover_minimum_interface_distance = 0.2**
(Any positive real number)
when crossing structures, this value is used to determine if any of the resulting bonds are too short. measured in angstroms. to be replaced by a distance ratio comparison like that in periodic structures.

**[mutation]**

The following keywords specify how structures are mutated

**forbidden_elements = None**
(Strings separated by whitespace or None. e.g. Au Ag H)
When mutating, these elements should be left in place.

**minimum_bond_length = 0.5**
(Any positive real number)
When mutating structures, this value is used to determine if any of the resulting bonds are too short. measured in angstroms. to be replaced by a distance ratio comparison like that in periodic structures.

**[lammps]**

The following keywords specify how LAMMPS is run

**path_to_lammps_executable = ???**
(Absolute path.)
Points to the lammps executable.

### [FHI-aims]

The following keywords specify how FHI-aims is run.

**path_to_aims_executable = ???**
(Absolute path)
points to the FHI-aims executable

**number_of_processors = 12**
(Integer)
determines how many processors aims will use in parallel. should be maximum number of processors on the node.

**initial_moment = hund**
(hund or custom)
used to specify the initial spin moment. if custom is used, the user-defined spin moments (in the user input geometries) will be used.

### [periodic]

The following keywords specify properties of periodic structure handling.

**periodic_system = False**
(Boolean)
Is the genetic algorithm expected to use periodic structures?

**periodic_model = model.in**
(Any string, must match a file name in the working directory.)
the model periodic structure is used to quickly define lattice vectors and the structure of a substrate if there is one.

**min_distance_ratio = 0.5**
(Positive real number)
this parameter and the elements' radii are used when comparing closeness across lattice cells

**lattice_vector_a = (10.0, 0.0, 0.0)**

**lattice_vector_b = (0.0, 10.0, 0.0)**

**lattice_vector_c = (0.0, 0.0, 10.0)**
(Python tuple of real numbers. length == 3)
defines one of the three lattice vectors used in periodic calculations

## 1.7 Modules

The modules are designed for easy interchangeability and freedom of implementation. Each module requires only four properties:

1. A function named main(...) exists which is called from the core algorithm.
2. The main function accepts particular arguments which differ from module to module.
3. The main function returns the necessary information required by the core algorithm.
4. The module does not alter the stored information used by the rest of the program.

The following lists the required input and output of each module as well as its purpose.

**Module Title**
Package: python_package_name
Arguments: arg1_type arg1_name, arg2_type arg2_name
Returns: return_type return_name
Effects: A description of the intended purpose of the module

**Initial Pool Filling**
Package: initial_pool
Arguments: int replica, StoicDict replica_stoichiometry
Returns: None
Effects: Randomly generates structures to be added to the initial pool. Scans the user-input structure folder for newly added structures. Performs a pre-relaxation desired. This module will be called at the beginning of each iteration and should do nothing if the initial pool is already filled. The structures should have the input reference -1 as they are conceptually a pre-cascade collection. Each structure added should have the property to_be_relaxed = True as each replica will search for initial pool structures to add to the collection of optimized structures.

**Random Structure Generation**
Package: random_gen
Arguments: StoicDict target_stoichiometry, float seed
Returns: Structure random_structure or False
Effects: Given a stoichiometry, a random structure is generated in order to fill the initial pool. A seed is also given in order to aid random coordinate generation. Depending on preference, a model structure may be read from a file in order to generate the random structure. However, this is not explicitly implemented and must be implemented by the user. See random_periodic.py for an example of model reading. If False is returned, iteration begins again.

**Structure Selection**
Package: selection
Arguments: dict<(StoicDict, int), StructureCollection> structure_supercollection, StoicDict target_stoic, int replica

Returns: list<Structure> structures_to_cross or False
Effects: Given a structure supercollection (simply a dictionary of all StructureCollections in memory), this module selects two or more structures to cross. It is responsible for stoichiometry choosing, fitness calculation, and making a weighted choice based on fitness. It is important to decide which level of cascade to choose from when selecting structures as the supercollection passed to this module contains every level including initial pool structures. One must also consider the cases where there are zero, one, or two structures present in the desired collection. If False is returned, iteration begins again.

**Crossover**
Package: crossover
Arguments: list<Structure> structures_to_cross, StoicDict target_stoic, int replica
Returns: Structure new_structure or False
Effects: Given a list of Structures and a target stoichiometry, this module will combine geometrical features of each structure in the list. The standard number of structures is two, but more could be crossed if desired. Various methods of crossover can be chosen within the module if desired though only one is currently implemented for cluster crossover. periodic crossover is also implemented and can be selected from the ui.conf file. If False is returned, iteration begins again.

**Mutation**
Package: mutation
Arguments: Structure structure_to_mutate, StoicDict target_stoic, int replica
Returns: Structure new_structure or False
Effects: A nowly crossed structure will always be passed through the mutation module so the decision to mutate or not must be made within the module. Furthermore, the method of mutation must be decided within the module (species switching, random translation, adding/subtracting atoms). These decision weights should be defined within the ui.conf file. If False is returned, iteration begins again.

**Relaxation**
Package: relaxation
Arguments: Structure input_structure, path working_dir, str control_in_string, int replica
Returns: Structure relaxed_structure or False
Effects: Given an unrelaxed input structure, this module will relax the structure or simply calculate its energy. This module is called multipme times throughout the cascade process with different control paramaters and therefore requires that the entire text of the control.in file is passed as a string. The working directory is the path pre-defined by the program where the calculation takes place. Relaxation may be made using any program or method. Currently implemented are FHI-aims relaxation and LAMMPS reaxff relaxation. See either of these modules for examples. If False is returned, iteration begins again.

**Structure Comparison**
Package: comparison
Arguments: Structure structure_to_compare, StructureCollection structure_collection, int replica
Returns: bool is_acceptable
Effects: Given a new Structure (relaxed or not) and its relative StructureCollection, this module will determine if it is an acceptable Structure to add to the StructureCollection. Common acceptability criteria are: acceptable fitness value (often energy) and structure geometry significantly different from existing geometries. Currently after an energy criteria is met, the new structure goes through a fine-grained comparison to existing structures with sufficienly similar energies and geometries. If these tests are passed True is returned. If not, False is returned and iteration begins again.

## 1.8 Core

!!! Clarify Core Logic

## 1.9 Examples

Where to find examples

Benchmarks

## 1.10 Known Bugs and Future Improvements

Bug description : estimated time

Allow for aims.out file to be kept alongside structure data : 1 day Perhaps consider switching to openbabel for better interface of structure handling. : 2 weeks

Closeness checking in rand_gen, mutation, and crossover modules should take element radius into consideration as in periodic structure checking : 1 hour

Actual element radii must be added to program, currently using dummy values. : 1 hour

Relaxation may be renamed to Optimization across whole program : 1 day

decide and implement how model structures are used and by which modules. random-gen for instance : 2 days

Add mutation options and implement a some decision model to choose which mutation is executed if any. : 3 days

# Chapter 2

# Parallel Genetic Algorithm Search

Comments and suggestions to:
newhouse@fhi-berlin.mpg.de
huynh@fhi-berlin.mpg.de
ghiringhelli@fhi-aims.berlin.mpg.de
levchenko@fhi-berlin.mpg.de.

A script based GA implementation is available. Part of the script is dependent on the particular batch-queuing system in use; with the distribution, we provide a solution that has been tested on linux machines with SGE batch-queuing system. Whereas the overall structure of the batch script would not change by changing the batch-queuing, few crucial lines might need intervention.

The script is available at "`/mnt/lxfs1/home/huynh/Free_Energy-GA`." Copy the entire folder into your working directory.

### 2.0.1 GA for atomic and periodic structures: the strategy

The theory and mechanics of Genetic Algorthims have been previously discussed at length by Johnston[1]. Here, we discuss how this Free Energy Genetic Algorithm compares to this discussion.

The goal of this Free Energy Genetic Algorithm is to find both the global configurational minimum and the global stoichiometric minimum, for given chemical potentials. That is, at some chemical potential, we determine the co-ordinates and composition of the most stable structure. We use parallized replicas to search this space, whose only interaction is accessing the common pool of accepted structures.

Instead of comparisons by total energy, we use an approximation for the free energy, that is:

$$\Delta F \sim E_f - E_{ref} - \sum_{a=atoms} \Delta n_a * \mu_a$$

$\Delta n_a$ : change in number of atoms *a* in each structure

$\mu_a$ : chemical potential for atom *a*
$E_f$ : total energy for final structure
$E_{ref}$ : total energy for initial (reference) structure

As a result of this approximation for free energy, the package includes a way to insert Temperatures and Pressures to obtain the full Gibbs Free Energy expression. This is done via the cascade method, which is explained more in the '`cascade`' section of the manual.

Fitness is a value assigned to each accepted structure, and correlates with the quality of the structure with respect to finding global minimums. A higher fitness corresponds to a higher quality structure. In this case, we are interested in finding the Global Minimum in Free Energy, which varies as chemical potential varies. Thus we split the search in chemical potential space amongst the replicas. As a result, each replica separately determines the quality of each structure.

Crossover is an operation between two structures selected by fitness. Higher structures have a higher probability to be selected for crossover. We use these two parent structures to form a child structure, with the hopes that positive genes and attributes from the parents are passed down to the children.

Mutation is used for two purposes here. The first is an attempt to avoid being trapped in a local minimum. As a result of continual crossover, the same genetic material is mixed and so it is possible that the pool becomes stale as no new structures are being added. Cue mutation which introduces structures with new attributes. This is done by switching atom species, or rotating a set of atoms within the structure.

The other use for mutation is to increase the pool of stoichiometry that we may search, by allowing mutation to change the number of atoms available to be searched. We do not let crossover change the stoichiometry so we can keep its role strictly to searching for configurational minima.

### 2.0.2 How to run

Here is a list of instructions once the user determines the molecules in the environment, and the conditions of pressure for each molecule and temperature of the system. The instructions denoted with a * are for users who wish to use the cascade function. Each instruction will have more detail in the corresponding `user_input.in` settings section.

- 1*: Determine T and p range for each molecule in the environment and edit the '`Generate Environment Settings`' section. Create a data file called '`environment_conditions.in`' with these values using the command, "`perl ./run/generate_conditions.plx`."
- 2*: Add the desired cascade functionals in '`./control/`'. Please note that in order for chemical potentials to be calculated, at least one of the cascading steps must be the vibration calculation, in order to obtain vibration frequencies of the molecule. Change the relavant sections in 'Cascade Settings.' In order to get

an accurate value for the Gibbs Free Energy of Formation, these functionals and settings should be kept the same when calculating for chemical potential of the environment and calculating Gibbs Free Energies for structures the GA produces.

- 3*: Once the user is satisfied with the generated temperatures and pressures, put environment geometries in the `'./higher_structures_copy/'` directory in the geometry.in format. The file names should read as '`./higher_structures_copy/`*`name`*" where name is the same used in `mu_header` under the `Generate Environment Settings`' section.
  e.g., if one has '`mu_H2O`' listed as a mu_header, then one should enter create a geometry file containing a water molecule called '`./higher_structures_copy/H2O`.'
  Do not add any '_' to the atom names here.

  Then, submit jobs_higher via '`qsub jobs_higher`' to put the environment molecules through the cascade process.

- 4*: Once all the environment geometries are completed the cascade process, then run the script "`perl ./run/chemical_potential.plx`" to create a file '`environment_conditions_chemical.in`' with the filled in chemical potentials.

- 5: Using the chemical potentials from step 4* or using the user's own chemical potentials, edit the '`Replicant Settings`' section for each molecule in the environment.

- 6: Edit the settings in '`General Settings`', '`Cluster/Surface Specific Settings`', '`General Mutation Settings`', and '`User Specific Settings`'. The guide for each section is below.

- 7: Add the environment geometries according to the '`./environment_geometry/`' section of the form '`./environment_geometry/H2O/geometry.in` (for a water molecule.) If no '`./control/settings_environment.dat`' file is provided, the geometries will not be relaxed. These environment geometries are used to add into structures via the mutation method.

- 8: Add initial structures into '`initial_pool`' in the geometry.in format. Add control.in settings for the GA as per the '`./control/`' section below.

- 9: To change the number of processors used by each replicant, please modify '`./jobs`' file.

- 10: To run, type "`perl ./master.plx`" or the command "`perl ./master.plx reset`" to delete files created by previous runs and start a new run. May add more replicants by submitting the "`./jobs`" file. If the cascade option is enabled, the cascading will start automatically.

- 11: During operation of the GA, one can view the progress of the structure creations with the files listed in the '`./progress/`' section. Once the GA has run for sufficient time, or the goal number of structures has been created, ensure all files in the '`./progress/`' directory are up to date. (They may not be because a replicant could have stopped running due to limit in run time and other replicants may have created more structures.) To update them, run "`./run/update.plx`." It will provide a list of the best structures in each stoichiometry existent in the system in the file '`master_geometry.in`.' Phase diagrams can now be plot according to the formula:

$$\Delta F \sim E_f - E_{ref} - \sum_{a=atoms} \Delta n_a * \mu_a$$

- 12*: Once all structures in '`higher_structures_copy`' have undergone the cascade, one can run "`perl ./run/Gibbs_Formation_Energy.plx` *`reference-structure`*," where *reference-structure* is the reference structure. This will use the existing file '`environment_conditions_chemical.in`,' and calculate the Gibbs Energy of Formation, according to the formula:

$$\Delta G(T, \mu_i) = G_{\text{new}}(T) - G_{\text{ref}}(T) - x\mu_i(T, p_i)$$

  One can use gnuplot or other plotting software to plot a phase diagram using the created files '`./higher_structures/structure_`$k$`/Gibbs_Energy_Formation.dat`.'

### 2.0.3 User input options

`user_input.in`: contains options available to the user. They are listed below:

**General Settings:**

- surface/cluster: determines whether the system is periodic or is a cluster
- minimum_bond_length: the bond lengths in each structure must be at least this far apart (in Å).
- number_of_children: how many replicas to send to the cluster. Can submit more replicas during run by submitting the "`./jobs`" file.
- number_of_structures: how many structures to be accepted before termination of program.
- control.in_species:default/custom
  Controls which basis sets for atom species are used in control.in files.
  - basic: uses default light/tight/really_tight settings as specified by user.
  - custom: for each atom in system, create files called `./control/`*`atomname`*`_main`, where *atomname* is the atom species.
    e.g. for a Hydrogen atom, create a file called `./control/H_main`
- light/tight/really_tight: if control.in_species setting is basic, determines which basis set to use.
- aims.out:keep/delete
  determine whether to keep or delete the aims.out files after energy is calculated.

- existence_minimum: to check if a structure already exists in the pool, first, all atom-atom distances are calculated for the current structure and stored as array. We compare with all structures with the same number of atoms by sorting the array and comparing it with the arrays from found structures, by subtracting one array from the other. With this new subtraction array, we sum up the elements and if this sum is greater than existence_minimum * length_of_array (the latter equal to the number of pairs), we have two unique structures.
- crossover_stoic:default/selective/parent
  Determine which stoichiometries the crossover process may accept.
  - default: stoichiometries already existent in the pool are accepted.
  - selective: stoichiometries in the top *leading_number* (see replicant settings) according to the replicant are accepted. Rankings are determined using fitnesses based on the free energy. This option promotes focused searches.
  - parent: stoichiometries shared by one of the two parent structures are accepted. This option promotes passing of genes.
- initial_moment:hund/custom
  Determine whether the moments of atoms will obey hund's rule, or are user-specified. Ensure this decision is reflected in the control.in settings.
  - hund: atoms will obey hund's rule
  - custom: the initial moment of atoms for structures in `initial_pool` must be specified by adding the line "initial_moment *moment*" after each atom, where *moment* is the moment of the atom, e.g.,
    ```
    atom     0.0     0.0     0.0     H
    initial_moment     0.0
    ```

**Cluster-Specific Settings:**

- crossover:default/percentage
  Determine the method of crossover
  - default: $z = 0$ plane is used to cut
  - percentage: the parent structure with higher fitness has a higher probability to have more atoms taken
- add_percent_box: lengths for box to add atoms will be add_percent_box times the distance of farthest atom from origin.
- prob_mutation1: if we do mutate, the probability that we remove at most remove_rate atoms.
- prob_mutation2: if we do mutate, the probability that we add at most adding_rate atoms from environment.
- prob_mutation3: if we do mutate, the probability that we exchange atom species within the system.

- prob_mutation4: if we do mutate, the probability that we exchange a pair atoms of different species.

**Surface-Specific Settings:**

- fixed_z-max: the maximum z value for which the atoms are fixed. Ensure that each atom below this z-value has the keyword `constrain_relaxation      .true.`
- prob_mutation5: if we do mutate, the probability that we remove at most max_remove_rate atoms.
- prob_mutation6: if we do mutate, the probability that we add at most max_adding_rate atoms.
- prob_mutation7: if we do mutate, the probability that we exchange a pair atoms of different species, both of which have $z >$ fixed_z-max.
- add_percentage_below: to add atoms, minimum height is add_below times the distance (maxz - fixedz) below the maximum z
- add_percentage_above: to add atoms, maximum height is add_above times the distance (maxz - fixedz) above the maximum z

**General Mutation Settings:**

- probability_of_mutation: after a crossover structure is created, the probability that we mutate a structure in the pool.
- forbidden_elements: elements which may not be removed by removal mutation.
- max_adding_rate: when adding atoms, will add up to this many.
- max_removal_rate: when removing atoms, will remove up to this many.
- environment_components: the chemical potentials atoms/molecule
  e.g. to have a chemical potential in terms of $H_2O$, add H_2:O_1
  and a chemical potential in terms of $H_2$, add H_2
- environment_elements: the elements that may be exchanged with the environment e.g. in the above example, the line should read `enviroment_elements      O  H`

**User Specific Settings:**

- BIN_DIRECTORY: specify the bin directory
- aims_file: specify the aims file you wish to execute

**Generating Environment Settings:**

For users who wish to explore the relationship between values of (T,p) and their corresponding chemical potentials. These options do not affect the GA directly, but instead provides chemical potentials for regions of temperature and pressure that the user may want to explore. These chemical potentials can be entered in the section `Replicant Settings`.
Instructions:

1: After configuring the settings for this section below, run the perl script "`perl ./run/generate_conditions.plx`." This will create a file '`environment_conditions.in`' containing the values of (T,p) for the chemical potential to be calculated. Review this file to ensure your settings below produced temperatures and pressure that you want to explore.

- constant_conditions:none/T/p_*atomname*
  Insert conditions to remain constant.
- constant_values: In the same order as above, insert values for each constant value in [K] or [atm] where applicable. If none was selected, enter '`n/a`'.
- variable_conditions:none/T/p_*atomname*
  Insert 1 or 2 conditions which one would like to examine the behaviour of the system over a range.
- scale_1:linear/log
  For the first variable condition, decide whether the range is over a linear or a logarithmic scale.
- minimum_1: Provide the lower bound for the range to be explored. [K/atm]
- maximum_1: Provide the upper bound for the range to be explored. [K/atm]
- divisions_1: If log scale is chosen, determines how many ticks of equal spacing will exist inbetween `VALUE` and `VALUE*log_base_1`.
- log_base_1: If log scale is chosen, determines the logarithmic base.
- delta_1: If linear scale is chosen, determines the number of ticks will exist, which will be of the form `(VALUE, VALUE + delta_1, VALUE + 2*delta_1, ..., maximum_1)`
- If one has a second variable condition, fill in the appropriate values for *_2.

2: Adjust the settings below and run the script "`perl ./run/chemical_potential.plx`" to fill in the chemical potentials for each element specified in `mu_header` option. These will be in the file '`environment_conditions_chemical.in`.' The user should note that the formula used to calculate these chemical potentials is:

$$\mu(T,p) = -\frac{3}{2}k_BT\ln\left(\frac{2\pi m k_BT}{h^2}\right) + k_BT\ln p - k_BT\ln\left(\frac{8\pi^2 I_A k_BT}{h^2}\right) + k_BT\ln\sigma$$
$$+\sum_i\left[\frac{hv_i}{2} + k_BT\ln\left(1-e^{-\frac{hv_i}{k_BT}}\right)\right] + E^{DFT} - k_BT\ln v_0$$

If the user wishes to use their own values for chemical potentials, they will have to write a script to enter their own values in.

- mu_header: Determines which atoms or molecules chemical potentials are to be calculated with the formula above. They should be of the form mu_*atomname*. The *atomname* must be the same as that of `./higher_structures_copy/`*`ATOMNAME/`* and should not contain any '_'.
- percentage_mu: Determines the percentage of the calculated chemical potential to that reported in '`environment_conditions_chemical.in`' in the same order as mu_*atomname* above. I.e., if the molecule O$_2$ has been cascaded to calculate the chemical potential, and the user would like to express his chemical potential as $\mu_O = \frac{1}{2}\mu_{O2}$, then one would enter 0.5 here.

3: Input a subset of the chemical potentials found into the `Replicant Settings` section, corresponding to the user's desired (T,p) values

**Replicant Settings:**

Note: *atomname* is referring to the section `environment_components`. Please ensure these match when naming chemical potentials.

- reference_*atomname*: specify chemical potential reference, $\mu_{\text{ref}}$, for *atomname* (in eV). One can set absolute chemical potentials below by setting this value to 0. e.g., specify "`reference_O_1       -2044.605302877255`" to set the chemical potential reference of oxygen to half the energy of an oxygen moleucle.
- chemical_potential_value:basic/custom
  Represents the change in chemical potential, $\Delta\mu$ from the reference (in eV)
  - basic: value set at basic_chemical_potential_*atomname* will be $\Delta\mu_{atomname}$ for all replicants
    e.g. if one wanted to set $\Delta\mu_{\text{H}_2\text{O}} = -1.0$eV for all replicants, one would enter "`basic_chemical_potential_He_1       -1.0"`
  - custom: value set at *child*_chemical_potential_*atomname* will be $\Delta\mu_{atomname}$ for *child*$^{th}$ replicant
    e.g. if one has an 8th replicant and wanted to set $\Delta\mu_{\text{H}_2\text{O}} = -1.0$eV for this replicant, one would enter "`8_chemical_potential_H_2:O_1       -1.0"`

- leading_number:basic/custom
  Only applicable when using the selective crossover method
  - basic: value set at basic_leading_number will be value for all children
  - custom: value set at *child*_leading_number will be value for the *child*$^{th}$ replicant
- basic_chemical_potential_*atomname*: change in chemical potential for *atomname*, for all replicants. (Only applies if chemical_potential_value option is basic)
- basic_leading_number: selective crossover will take accept the top this number of stoichiometries. Applies to all replicants. (Only applies if leading_number option is basic)
- *child*_chemical_potential_*atomname*: change in chemical potential for *atomname*, for the *child*$^{th}$ replicant. (Only applies if chemical_potential_value option is custom)
- *child*_leading_number: selective crossover will take accept the top this number of stoichiometries. Applies to the *child*$^{th}$ replicant. (Only applies if leading_number option is custom)

**Cascade Settings:**

If the user wishes to obtain more accurate comparisons between structures, they may choose to use the cascade function. With this, the GA selects the most stable structures and optimizes their geometries and calculates their energies, using functionals specified by the user. The structures chosen for this cascade treatment are the more stable structures for each stoichiometry, as well as the structures in the top `higher_threshold` [eV] for each GA-replicant. Only structures that have undergone cascade with the vibration option can have their Gibbs Formation Energy calculated according to:

$$\begin{aligned}
G^{(\text{linear})}(T) = & -\frac{3}{2}k_BT\ln\left(\frac{2\pi m k_BT}{h^2}\right) - k_BT\ln\left(\frac{8\pi^2 I_A k_BT}{h^2}\right) + k_BT\ln\sigma \\
& + \sum_i\left[\frac{hv_i}{2} + k_BT\ln\left(1-e^{-\frac{hv_i}{k_BT}}\right)\right] + E^{DFT} - k_BT\ln v_0 \\
G^{(\text{non-linear})}(T) = & -\frac{3}{2}k_BT\ln\left(\frac{2\pi m k_BT}{h^2}\right) - k_BT\ln\left[8\pi^2\left(\frac{2\pi k_BT}{h^2}\right)^{\frac{3}{2}}\right] - \frac{1}{2}k_BT\ln(I_AI_BI_C) \\
& + k_BT\ln\sigma + \sum_i\left[\frac{hv_i}{2} + k_BT\ln\left(1-e^{-\frac{hv_i}{k_BT}}\right)\right] + E^{DFT} - k_BT\ln v_0
\end{aligned}$$

Specific Instructions for Cascade: For each step in the cascade, one has to provide settings files, as well as basis sets for each atom in the system. The settings files must be of the form '`./control/settings_`***`functional`***' and the species files are to be of the form '`./control/Mg_`***`functional`***.'

Please refer to the sample template which is for systems of Mg, O and H found in the `./control/` directory:
First, it optimizes the geometry further with the PBE functional and tight settings, then calculates the vibration modes of the structure with PBE and tight settings and finally calculates the energy with PBE0 and tight settings.
To submit additional jobs, (i.e., if the GA is not running, but user would like to cascade more structures), use the command '`qsub jobs_higher`'.

- cascade_start: determines the number of structures that are created before structures are considered for the cascade process. Set to 0 for no cascade
- cascade_replicants: specifies the number of jobs to submit dedicated to cascading the structures.
- higher_threshold: for each GA-replicant, structures with a free energy value in the top `higher_threshold` eV will be selected.
- control_order: determines the order of settings files in the folder `./control/` with which to run aims. For example, if one has control settings '`settings_PBE.dat settings_vibration.dat settings_PBE0.dat`,' one would enter '`PBE vibration PBE0`."
- relaxation_structure: specify which functional(s) where the calculated relaxed geometry should be updated in geometry.in.
- energy_structure: specify which functional from which the total energy value should be taken.

### 2.0.4 ./control/

Directory containing control.in settings.

`settings_main.dat`: the control.in settings which AIMS will use for system structures

`settings_environment.dat`: the control.in settings which AIMS will use to relax environment atoms and molecules

`settings_`*`functional`*: the control.in settings which AIMS will use for the *functional* stage of the cascade, if enabled

*`atomname_functional`*: if using custom control.in species option (see general settings), contains the basis set for the species *atomname* to be used in control.in files
e.g., `./control/H_main` contains the basis set for the Hydrogen atom for the GA (only if using custom control.in species option).
e.g., `./control/O_PBE` contains the basis set for the Oxygen atom for the cascade portion using the settings '`./control/settings_PBE.dat`.'

### 2.0.5 ./environment_geometry/

If system exchanges atoms with environment, add geometry files here containing environment atoms or molecules.
e.g., for a system whose environment contains Hydrogen, create a file
`./environment_geometry/H/geometry.in` containing
"`atom 0.0 0.0 0.0 H`"
The geometries will be relaxed according to the control.in settings in
'`./control/settings_environment.dat`.'

### 2.0.6 ./progress/

Contains information about comparisons between structures.

`etot.dat`: lists each structure with its total energy in eV and stoichiometry

`fitness.dat`***`child`***: lists each structure with its fitness and stoichiometry for the *child*$^{th}$ replicant.
Fitness is calculated by scaling the free energy from $[0, 1]$, and using the function $f = \exp(-3 * F)$ where $F$ is the scaled free energy.

`getot.dat`***`child`***: lists each structure with its difference in free energy and stoichiometry for *child*$^{th}$ replicant.
Difference in free energy's reference is always the first initial_pool structure to have its energy converged

`leading_structures.dat`***`child`***: lists the leading structure for all stoichiometries present in the pool for *child*$^{th}$ replicant, only meaningful with non-constant stoics.

### 2.0.7 Important Files/Directories

`result.out & jobs.o*`: print statements to show the progress of the program's operation.

`./structures/`: directory with all structures.

`./structures/structure_i/`: contains `aims.out`, `geometry.in` and `control.in` files. Total energy is also in `energy.dat`.

`./higher_structures/structure_i/`: contains vibration calculations, energy values and optimized geometry from cascade treatment (if enabled).

`./user_structures/`: directory to add new structures and hence (optionally) new stoichiometries. Files must be in `geometry.in` format. It appears only during operation of program

### 2.0.8 Reference

[1] Johnston, Roy L. "Evolving Better Nanoparticles: Genetic Algorithms for Optimising Cluster Geometries." Dalton Transactions 22 (2003): 4193-207. Print.

## Following pages: Index

(this page inserted to enforce proper hyperlink to index)