Module Interface Specification for Kaplan

Jen Garner

April 20, 2019

1 Revision History

Date	Version	Notes
November 20, 2018 (Tuesday)	1.0	Initial draft
November 26, 2018 (Monday)	1.1	Complete first draft
December 4, 2018 (Tuesday)	1.2	Update fitg module to remove the wrapper function and make most of the functions access routines instead of local functions

2 Symbols, Abbreviations and Acronyms

See https://github.com/PeaWagon/Kaplan/blob/master/docs/SRS/SRS.pdf Documentation.
cid = compound identification number (for https://pubchem.ncbi.nlm.nih.gov/ website)
vetee = private database repository on github

Contents

3 Introduction

The following document details the Module Interface Specifications (MIS) for Kaplan. This program is designed to search a potential energy space for a set of conformers for a given input molecule. The energy and RMSD are used to optimize dihedral angles, which can then be combined with an original geometry specification to determine an overall structure for a conformational isomer.

Complementary documents include the System Requirement Specifications (SRS) and Module Guide (MG). The full documentation and implementation can be found at https://github.com/PeaWagon/Kaplan.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$. Also, the PEP8 style guide from Python will be used for naming conventions.

The following table summarizes the primitive data types used by Kaplan.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
boolean	bool	True or False

The specification of Kaplan uses some derived data types: sequences, strings, tuples, lists, and dictionaries. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a fixed list of values, potentially of different types. Lists are similar to tuples, except that they can change in size and their entries can be modified. For strings, lists, and tuples, the index can be used to retrieve a value at a certain location. Indexing starts at 0 and continues until the length of the item minus one (example: for a list my_list = [1,2,3], my_list[1] returns 2). A slice of these data types affords a subsection of the original data (example: given a string s = "kaplan", s[2:4] gives "pl"). Notice that the slice's second value is a non-inclusive bound. A dictionary is a dynamic set of key-value pairs, where the keys and the values can be modified and of any type. A dictionary value is accessed by calling its key, as in dictionary_name[key_name] = value.

Kaplan uses three special objects called Pmem, Ring, and Parser. These objects have methods that are described in 13, 12, and 15 respectively. The Python NoneType type object is also used.

Here is a table to summarize the derived data types:

Data Type	Notation	Description
population member	Pmem	an object used by Kaplan to represent potential solutions to the conformer search/optimization problem
ring	Ring	an object used by Kaplan to store Pmem objects and define how they are removed, added, and updated
parser	Parser	a Vetee object used by Kaplan to represent the molecular geometry, its energy calculations, and input/output parameters; also inherited classes include: Xyz, Com, Glog, Structure
string	str	a string is a list of characters (as implemented in Python)
list	list or [ ]	a Python list (doubly-linked)
dictionary	dict or {}	a Python dictionary that has key value pairs
NoneType	None	empty data type

In addition, Kaplan uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification. There is one generator function in this program, which uses the yield keyword instead of the return keyword. Every time a generator is called, it returns the next value in what is usually a for loop.

Note that obvious errors (such as missing inputs) that are handled by the Python interpreter are not listed under the exceptions in any of the Kaplan modules. [good —SS]

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	GA Input Molecule Input GA Control Fit_G Tournament Crossover & Mutation Ring Pmem Output
Software Decision Module	Geometry Energies RMSD

Table 1: Module Hierarchy

6 MIS of GA Input

[Right now the link does not open the SRS document. Not sure if that was supposed to happen. —JG] [There are ways to get the external links to work, but don't worry about it; it isn't worth your time to fiddle with that right now. —SS]

The purpose of this module is to provide a utility for reading and verifying input related to the genetic algorithm (GA). There are two main functions: read_ga_input and verify_ga_input. The first function opens a data file (.txt file) with the following format:

```
num_mevs = 1000
num_slots = 100
num_filled = 20
num_geoms = 3
num_atoms = 10
t_size = 7
num_muts = 3
num_swaps = 1
pmem_dist = 5
fit_form = 0
coef_energy = 0.5
coef_rmsd = 0.5
```

These values are read into a Python dictionary, called ga_input_dict. The order of the inputs does not matter, but Kaplan will throw an error if one of the keys is missing. This dictionary is then passed to the second function, which checks that the values are correct and that all keys have been given. From the SRS document (see SRS Section 2.2), [I like that you have external references, but they only work if the SRS document is compiled. It would be nice if you had a makefile that made all of the documents, like the one in the Blank Project example in our repo. —SS] n_G and n_a are represented here as num_geoms and num_atoms respectively. Also, coef_energy and coef_rmsd are C_{E} and C_{RMSD} from the SRS. All keys are case insensitive. The values are case insensitive except for the SMILES string (if chosen struct_type is smiles), which is case sensitive by definition.

[I may have to check if SMILES strings are case sensitive for the programs I am parsing them with. —JG] [okay —SS]

[Turns out that SMILES strings are case sensitive, since lowercase indicates an aromatic atom. I was able to catch this problem during my testing and I have addressed the issue in the code. —JG]

6.1 Module

ga_input

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

NUM_GA_ARGS := 12

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
read_ga_input	str	dict	FileNotFoundError
verify_ga_input	dict	None	ValueError

6.4 Semantics

6.4.1 State Variables

num_args := for each line in the ga_input_file $+(x : list | key \in x \wedge value \in x \wedge length(x) = 2 : 1)$
[what list is this based on? —SS]
ga_input_dict, which is a dictionary that contains:

- num_mevs := $\mathbb{N}$
- num_geoms := $\mathbb{N}$
- num_atoms := $\{x \in \mathbb{N} : x > 3\}$
- num_slots := $\{x \in \mathbb{N} : x \geq \text{num\_filled}\}$
- num_filled := $\{x \in \mathbb{N} : x \leq \text{num\_slots}\}$
- num_muts := $\{0 \vee x \in \mathbb{N} : \text{num\_atoms} \geq x \geq 0\}$
- num_swaps := $\{0 \vee x \in \mathbb{N} : \text{num\_geoms} \geq x \geq 0\}$
- t_size := $\{x \in \mathbb{N} : 2 \geq x \leq \text{num\_filled}\}$
- pmem_dist := $\{0 \vee x \in \mathbb{N} : x \geq 0\}$
- fit_form := $\{0 \vee x \in \mathbb{N} : x \geq 0\}$
- coef_energy: $\mathbb{R}$

- coef_rmsd: $\mathbb{R}$

[You don't actually have state variables for the above, since you are outputting this information in a dictionary. What you have done is partially defined the abstract data type for your dictionary. I don't understand why you need a dictionary? I don't see what the key value is? Can't you just have input data have the state variables that you have mentioned and skip the idea of outputting a dictionary? The input module would be available to any module that needed these inputs and, if it is implemented as a singleton object, you don't even need to pass an object. —SS]

6.4.2 Environment Variables

ga_input_file: str representing the file that exists in the working directory (optionally includes a prepended path).

6.4.3 Assumptions

This module is responsible for all type checking and no errors will come from incorrect passing of variables (except input related to the molecule - covered in 7).

6.4.4 Access Routine Semantics

read_ga_input(ga_input_file):

- transition: open ga_input_file and read its contents.
- output: dictionary (ga_input_dict) that contains the values listed in State Variables.
- exception: FileNotFoundError := ga_input_file $\notin$ current working directory.

verify_ga_input(ga_input_dict):

- transition: None
- output: None
- exception: ValueError
  - 4 >num_atoms
  - num_filled >num_slots
  - num_swaps >num_geoms
  - t_size >num_filled $\vee$ t_size <2
  - not an integer type (for all except coef_energy and coef_rmsd, which should be floats)
  - missing key/unknown key
  - too many keys (i.e. repeated keys), where num_args $\neq$ NUM_GA_ARGS

6.4.5 Local Functions

None

[For ease of reference, it is nice if there is a newpage before each module. —SS]

7 MIS of Molecule Input

The purpose of this module is to provide a utility for reading and verifying input related to the molecule, including its structure and energy calculations. There are two main functions: read_mol_input and verify_mol_input. The first function opens a data file (.txt file) with the following format:

```
qcm = hartree-fock
basis = aug-cc-pvtz
struct_input = C=CC=C
struct_type = smiles
prog = psi4
charge = 0
multip = 1
```

These values are read into a Python dictionary, called mol_input_dict. The order of the inputs does not matter, but Kaplan will throw an error if one of the keys is missing. This dictionary is then passed to the second function, which checks that the values are correct and that all keys have been given. To verify the molecular input, Vetee's Parser object is constructed using the mol_input_dict (geometry module, Section 15). The mol_input module calls the energy module (Section 16) to run a calculation on the input molecule. If this calculation converges, then the manipulation of the dihedral angles are more likely to afford calculations that converge. After this final verification, the Parser object is passed back to the gac module, and eventually gets used by the pmem module (Section 13). From the SRS document (see SRS Section 2.2), QCM and BS are represented here as qcm and basis respectively. All keys and string values are case insensitive.

7.1 Module

mol_input

7.2 Uses

geometry (Section 15), energy (Section 16)

7.3 Syntax

7.3.1 Exported Constants

NUM_MOL_ARGS := 7

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
read_mol_input	str	dict	FileNotFoundError
verify_mol_input	dict	Parser	ValueError

7.4 Semantics

7.4.1 State Variables

num_args := for each line in the mol_input_file $+(x : list | key \in x \wedge value \in x \wedge length(x) = 2 : 1)$

mol_input_dict, which is a dictionary that contains:

- qcm := str $\in$ available methods given prog
- basis := str $\in$ available basis sets given prog and molecule
- struct_input := $\{x : str | x = file \vee x = SMILES \vee x = name \vee x = cid : x\}$
- struct_type := $str \in \{$"*smiles*", "*xyz*", "*com*", "*glog*", "*name*", "*cid*"$\}$
- prog := $str \in \{$"*psi4*"$\}$
- charge := $\mathbb{Z}$
- multip := $\mathbb{N}$

[The same comment applies about the dictionary as for the previous module. Rather than using strings for the structure types, you could considering using an enumerated type. —SS]

7.4.2 Environment Variables

mol_input_file: str representing the file that exists in the working directory (optionally includes a prepended path).

7.4.3 Assumptions

As with 6, other modules that use and exchange the state variables found in this module will not raise errors related to the type of input.

7.4.4 Access Routine Semantics

read_mol_input(mol_input_file):

- transition: open mol_input_file and read its contents.
- output: dictionary (mol_input_dict) that contains the values listed in State Variables.
- exception: FileNotFoundError := mol_input_file $\notin$ current working directory.

verify_mol_input(mol_input_dict):

- transition: None
- output: Parser
- exception: ValueError
  - qcm $\notin$ prog
  - basis $\notin$ prog $\vee$ basis unavailable for molecule
  - unable to parse SMILES string, name, cid, or input file
  - struct_type not available
  - not a string type (for all except charge $\mathbb{Z}$ and multip $\mathbb{N}$)
  - missing key/unknown key
  - too many keys (i.e. repeated keys), where num_args $\neq$ NUM_MOL_ARGS

7.4.5 Local Functions

None

8 MIS of GA Control

This module is responsible for running the GA using the given inputs. The general format of the algorithm is as follows:

1. Read in and verify ga_input_file (6).
2. Read in and verify mol_input_file (7). This step includes a check of the initial geometry, QCM, and BS for convergence (16), and generating a Parser object (15).
3. Generate a Ring object (12).
4. Fill the Ring with Pmem objects according to the num_filled input variable (13).
5. Iterate over the num_mevs input variable, and run a tournament on the Ring according to the t_size variable (10).
6. Return the output as per the output module specifications (14).

8.1 Module

gac

8.2 Uses

ga_input (Section 6), mol_input (Section 7), output (Section 14), ring (Section 12), tournament (Section 10)

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

[Not sure if I should list the exceptions raised by imported modules here? For example, reading the input may give an error, but this error is not explicitly raised by the gac module. Do I still have to list it here? —JG] [No, only listed exceptions that are the responsibility of this module to raise. —SS]

Name	In	Out	Exceptions
run_kaplan	str, str	None	None

8.4 Semantics

8.4.1 State Variables

[I don't actually need to store the mol_input_dict variable here since its information will be completely contained in the Parser object. Mostly I am leaving this note here as a reminder to update the Parser object with the prog attribute. This update would also mean consolidating the function calls read and verify mol input. Not sure if it is a good idea to have these separated in gac? —JG]

- ga_input_dict : dict (see 6)
- mol_input_dict : dict (see 7)
- parser : Parser (see 15)
- ring : Ring (see 12)
- mev : $\{x : \mathbb{Z} | \text{mum\_mevs} > x \geq 0 : x\}$

[I don't entirely understand your design. The confusion about the role of state variables is muddying the water for me. I can tell you have done a great deal of work and have thought deeply about the design. My guess is that you have thought about the design in Python and

you are trying to document your Python design using the 741 MIS. I think this has led to a more complicated MIS document than necessary —SS]

[I've been trying to think of practical advice that can help the documentation, but not consume too much of your time. What I've come up with is that you should replace each of your dictionaries with newly defined types. The new types will be tuples, in the Hoffmann and Strooper sense, but they will be implemented by dictionaries. A dictionary gives a set of key value pairs. A tuple is a set of field names and values. This would let you introduce the new types using the H&S notation, but you don't have to document them as ADTs because the implementation as dictionaries is so straightforward. You can define all of your new types in section 4, or in a type defining module. —SS]

8.4.2 Environment Variables

None

8.4.3 Assumptions

This is the main control unit for the program; the user will write their own input files and only need to access this module to complete their task.

8.4.4 Access Routine Semantics

run_kaplan(ga_input_file, mol_input_file):

- transition: The set of conformers with large negative energy and high RMSD are produced and passed to the output module (14). [if you are passing something, it isn't a state variable. —SS]
- output: None
- exception: None

[I think you have a misunderstanding about the state variables. The fields of an object are specified as state variables. If you have state variables, you should have state transitions that show how the values are changed. —SS]

8.4.5 Local Functions

None [Since the run_kaplan function is only used by gac, should I put it here in local functions? Technically the user will have to import it somewhere to use it. —JG]

9 MIS of Fit_G

The purpose of this module is to calculate the fitness of the Pmem object. In ga_input_file, the user specifies the fit_form (the formula number to use for calculating fitness), the coefficients

for the energy and RMSD terms, the method (QCM), and the basis set (BS). These values are used here to assign a fitness to a pmem. A change of dihedral angles may make it impossible for an energy calculation to converge; in this case the energy will be set to zero, but the RMSD value will most likely be high for the set of conformers. The contribution of the RMSD value to the fitness should therefore be smaller than the energetic component (otherwise the user may end up with multiple non-convergent geometries with high RMSD).

The Fit_G module works as follows:

1. The ring calls its set_fitness method, which calls the sum_energies and sum_rmsds functions, and passes the outputs of these functions to the calc_fitness function.

2. The output of the calc_fitness function depends on the chosen fit_form, which for now is only allowed to be 0. The resulting floating point is returned to the ring.

9.1 Module

fitg

9.2 Uses

energy (Section 16), rmsd (Section 17)

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

The sum_energies function should raise a warning if an energy calculation did not converge.

Name	In	Out	Exceptions
sum_energies	list(list(str,$\mathbb{R}$,$\mathbb{R}$,$\mathbb{R}$)), $\mathbb{Z}$, $\mathbb{N}$, str, str	$\mathbb{R}$	None
sum_rmsd	list(list(str,$\mathbb{R}$,$\mathbb{R}$,$\mathbb{R}$))	$\mathbb{R}$	None
calc_fitness	$\mathbb{Z}$, $\mathbb{R}$, $\mathbb{R}$, $\mathbb{R}$, $\mathbb{R}$	$\mathbb{R}$	None

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

New fitness functions will be added. Each new fitness functions should be incrementally labelled and added to the calc_fitness function. Any new fitness functions that are added should account for the possibility of divide by zero errors.

9.4.4 Access Routine Semantics

sum_energies(xyz_coords, charge, multip, method, basis):

- transition: None
- output: $out := +(x : \mathbb{R} | x =$ energy of xyz_coords with charge, multiplicity, QCM, BS : $|x|)$
- sum together the energies of the conformer geometries for one solution instance (one pmem).
- exception: None

sum_rmsds(xyz_coords):

- transition: None
- output: $out := +(x(i,j) : \mathbb{R} |$RMSD between coords i and j where i, j are indices $\forall i, j \in$ xyz_coords : $x)$
- determine possible pairs of conformers (with all_pairs_gen) and calculate their rmsd by using rmsd module.
- exception: None

calc_fitness(fit_form, sum_energy, coef_energy, sum_rmsd, coef_rmsd):

- transition: None
- output: when fit_form = 0, then $out := \{ << C_E, S_E, C_{\text{RMSD}}, S_{\text{RMSD}} >, y >: \mathbb{R} | y = C_E * S_E + C_{\text{RMSD}} * S_{\text{RMSD}} \}$
- exception: None (may complain if inputs have not been previously checked i.e. fit_form is not available)

9.4.5 Local Functions

$\mathbb{Z} \rightarrow$ tuple($\mathbb{Z}, \mathbb{Z}$)
all_pairs_gen(num_geoms):

- transition: increment iterators i and j in the generator function.
- output: $out := \{< i, j >: \mathbb{Z} | 0 \leq i \leq \text{num\_geoms} - 1 \wedge i + 1 \leq j \leq \text{num\_geoms}\}$
- exception: None
- the generator function, all_pairs_gen, will need to keep track of the i and j iteration values.
- num_pairs is the number of next() calls needed for the generator. $:= \{< y, n > | n = \text{num\_geoms} : \mathbb{N} \wedge y = n!/(2 * (n-2)!)\}$

10 MIS of Tournament

The tournament module selects t_size pmems from the ring for comparison. It ranks the pmems in order of increasing fitness, as calculated using the fitg module (9). Then, the two best pmems are chosen as "parents", and the mutations module (11) is used to generate two new "children" based on these parents. The ring module (12) then decides whether the children are added to the ring or not, and if old pmems are deleted to make room for the children.

10.1 Module

tournament

10.2 Uses

ring (Section 12), pmem (Section 13), mutations (Section 11)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
run_tournament	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Z}$, Ring	None	EmptyRingError
select_pmems*	$\mathbb{N}$, Ring	list($\mathbb{Z}$)	None
select_parents*	list, Ring	tuple($\mathbb{Z}$,$\mathbb{Z}$)	None

* local function

10.4 Semantics

10.4.1 State Variables

- selected_pmems := $\{x : \mathbb{N} | x \geq 0 \wedge \text{ring.pmems}[x] \neq None\}$
- parents := $\{x : \mathbb{N} | x \geq 0 \wedge \text{ring.pmems}[x] \neq None\}$
- parent1 := $[[D_1], [D_2], ..., [D_{n_G}]]$, where each D_i is a list of length num_atoms-3 of integers representing the dihedral angles
- parent2 := $[[D_1], [D_2], ..., [D_{n_G}]]$, where each D_i is a list of length num_atoms-3 of integers representing the dihedral angles
- children := $[[[D_1], [D_2], ..., [D_{n_G}]], [[D_1], [D_2], ..., [D_{n_G}]]]$, where each D_i is a list of length num_atoms-3 of integers representing the dihedral angles

10.4.2 Environment Variables

None

10.4.3 Assumptions

None [May have to think on this more. —JG]

10.4.4 Access Routine Semantics

run_tournament(t_size, num_muts, num_swaps, ring, current_mev):

- transition: new ring members may be added with birthday equal to the current_mev (increments ring.num_filled), and old pmems may be replaced with a new pmem.
- output: None
- exception: EmptyRingError when t_size > ring.num_filled

10.4.5 Local Functions

select_pmems(number, ring):

- transition: None
- output: selection, which is a list of length *number* of random ring indices containing a pmem; no pmem can appear twice in the selection.
- exception: None

select_parents(selected_pmems, ring):

- transition: None
- output: tuple of length 2 representing the ring indices of the pmems with the best fitness values out of the selected_pmems indices.
- exception: None

11 MIS of Crossover & Mutation

This module is used to generate new solution instances with which to generate new pmems (13) for the ring (12). It has two local functions, mutate and swap. Mutate takes in a list of lists representing dihedral angles for the molecule of interest. Then, a number of these dihedral angles are randomly changed. Swap takes in two of such list of lists and swaps a number of sublists between the two inputs. This module is called during a tournament (10). A wrapper function to call these two functions is called generate_children, which returns the new solution instances to the tournament.

11.1 Module

mutations

11.2 Uses

None

11.3 Syntax

11.3.1 Exported Constants

These two values restrict the dihedral angles that can be chosen.
MIN_VALUE : 0
MAX_VALUE : 360

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
generate_children	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$)), $\mathbb{Z}$, $\mathbb{Z}$	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$))	None
mutate*	list(list($\mathbb{Z}$)), $\mathbb{Z}$	list(list($\mathbb{Z}$))	None
swap*	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$)), $\mathbb{Z}$	list(list($\mathbb{Z}$)), list(list($\mathbb{Z}$))	None

* local function

11.4 Semantics

11.4.1 State Variables

- number of chosen swaps := $\{x : \mathbb{Z} | 0 \leq x \leq$ num_swaps from ga_input_dict$\}$
- number of chosen mutations := $\{x : \mathbb{Z} | 0 \leq x \leq$ num_muts from ga_input_dict$\}$

11.4.2 Environment Variables

None

11.4.3 Assumptions

None [Maybe good to reference SRS assumptions about inputs here? —JG]

11.4.4 Access Routine Semantics

generate_children(parent1, parent2, num_muts, num_swaps):

- transition: None
- output: two list of lists of integers between MIN_VALUE and MAX_VALUE representing dihedrals for two new population members. These will be used to create pmem objects.
- exception: None

11.4.5 Local Functions

mutate(dihedrals, num_muts):

- transition: locally update num_muts with a random number (min 0, max num_muts).
- output: a list of lists of integers between MIN_VALUE and MAX_VALUE representing dihedrals for one mutated population member.
- exception: None

swap(parent1, parent2, num_swaps):

- transition: locally update num_swaps with a random number (min 0, max num_swaps).
- output: two list of lists of integers between MIN_VALUE and MAX_VALUE representing dihedrals for two swapped population members.
- exception: None

12 MIS of Ring

The ring is the main data structure for Kaplan. It determines how the potential solutions to the conformer optimization program are organized. The constructor for the ring takes 5 arguments: num_geoms, num_atoms, num_slots, pmem_dist, and parser. The ring begins empty and can be filled with pmem objects by calling the ring.fill method. There is also the ring.update method, which takes 2 arguments: parent_index and child. This method is called during a tournament after the children have been generated. The update occurs as follows:

1. Select a random slot in the range [parent_index-pmem_dist, parent_index+pmem_dist+1] from the parent.
2. Compare the fitness value of the child with the fitness value of the current occupant.
3. If there is no current occupant, or if the child has fitness $\geq$ the current occupant, put the child in the slot.
4. Increment the num_filled attribute of the ring if an empty slot was filled.

The ring also uses the geometry module (Section 15) to generate a zmatrix as a string based on the pmem index of interest. Calling ring.calc_fitness will determine the fitness for a given pmem index.

12.1 Module

ring

12.2 Uses

pmem (Section 13), fitg (Section 9), geometry (Section 15)

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
__init__	$\mathbb{N}, \mathbb{N}, \mathbb{N}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z}, \mathbb{Z}$ Parser	None	None
set_fitness	$\mathbb{Z}$	None	ValueError
update	$\mathbb{Z}$, list(list[$\mathbb{Z}$])	None	None
fill	$\mathbb{N}, \mathbb{Z}$	None	RingOverflowError
RingEmptyError	-	-	-
RingOverflowError	-	-	-

[You should specify a constructor, not __init__. Your specification is very Pythoncentric. The goal should be to make it as language agnostic as possible. —SS]

12.4 Semantics

12.4.1 State Variables

- ring.num_geoms from ga_input_module
- ring.num_atoms from ga_input_module
- ring.pmem_dist from ga_input_module
- ring.fit_form from ga_input_module
- ring.coef_energy from ga_input_module
- ring.coef_rmsd from ga_input_module
- ring.parser from the geometry module
- ring.num_filled from ga_input_module; represents the number of pmems present in the ring (dynamic)
- ring.pmems is a list of pmem objects (pmem module) or NoneType objects (depends if slot is filled or empty)

12.4.2 Environment Variables

None

12.4.3 Assumptions

- the ring module will be written in such a way as to support the addition of extinction operators. These extinction operators delete segments and/or pmems with certain attributes from the ring.
- a pmem cannot be initialized without a call by the ring to evaluate its fitness. This evaluation will not be a wasted computation.
- The ring can be iterated and will not fall over when an index past zero or above the last slot is called (index wrapping).

12.4.4 Access Routine Semantics

__init__(num_geoms, num_atoms, num_slots, pmem_dist, fit_form, coef_energy, coef_rmsd, parser):

- transition: generate a ring object.
- output: None
- exception: None

update(parent_index, child):

- transition: generate a pmem with the child (the sets of dihedral angles), and calculate its fitness. Select a slot to place the child within [parent_index-pmem_dist, parent_index+pmem_dist+1]. If the slot is occupied, the child must have fitness that is no worse than the current occupant. Note: this will require wrapping for the Ring to ensure that an IndexError is not raised. Increment the num_filled attribute if a slot that was once empty is filled.
- output: None
- exception: None

fill(num_pmems, current_mev):

- transition: if there are no pmems in the ring (num_filled = 0), fill the ring with a contiguous segment of num_pmems pmems. If there are pmems in the ring, fill empty slots with new pmems until num_pmem pmems have been added. For each new pmem, calculate its fitness.
- output: None
- exception: RingOverflowError occurs when there is a request to add a set of pmems to the ring that the number of free slots does not accommodate.

12.4.5 Local Functions

set_fitness(pmem_index):

- transition: updates the pmem.fitness attribute by constructing a zmatrix using the geometry module and calling get_fitness from the fitg module.
- output: None
- exception: ValueError occurs if the slot is empty at pmem_index.

13 MIS of Pmem

This module is designed to hold the pmem data structure. A pmem is generated by the ring module. The pmem holds the dihedrals list of lists, which is what the Kaplan program is optimizing.

13.1 Module

pmem

13.2 Uses

None

13.3 Syntax

13.3.1 Exported Constants

These two values restrict the dihedral angles that can be chosen.
MIN_VALUE := 0
MAX_VALUE := 360

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
__init__	$\mathbb{Z}, \mathbb{N}, \mathbb{N}, \mathbb{Z}, list(list(\mathbb{Z}))*$	None	None

* the default value is None

13.4 Semantics

13.4.1 State Variables

- pmem.ring_loc := the ring index where the pmem is located.
- pmem.dihedrals := $[[D_1], [D_2], ..., [D_{n_G}]]$, where each D_i is a list of length num_atoms-3 of $\mathbb{Z}$ representing the dihedral angles. If the constructor is called without a given set of dihedrals, then the default will be to randomly fill in those values between MIN_VALUE and MAX_VALUE.
- pmem.fitness := float representing result of a fitg calculation for the pmem's dihedral angles when combined with the other geometry specifications in ring.parser.
- pmem.birthday := $\{x : \mathbb{Z} | num\_mevs > x \geq 0\}$ mating event for which the pmem was generated

13.4.2 Environment Variables

None

13.4.3 Assumptions

After a pmem object is generated, its fitness will be calculated.

13.4.4 Access Routine Semantics

__init__(ring_loc, num_geoms, num_atoms, current_mev, dihedrals=None):

- transition: generate a new pmem object.
- output: None
- exception: None

13.4.5 Local Functions

None

14 MIS of Output

The output module is called by the GA Control module (8) to produce output for the program. There are two main requirements: return the best set of conformer geometries with full geometry specifications and calculate their respective energies. Although it is not listed in the requirements, this module will most likely be upgraded to include some statistical measurements of the results.

14.1 Module

output

14.2 Uses

geometry (Section 15), ring (Section 12), pmem (Section 13)

14.3 Syntax

14.3.1 Exported Constants

OUTPUT_FORMAT = “xyz”

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
run_output	Ring	$\mathbb{R}, \mathbb{R}$, list($\mathbb{R}$), Ring	None*

*Need an exception in case the program does not have the permissions needed to write to the output directory.

14.4 Semantics

14.4.1 State Variables

- total_fit : the total sum of all fitness values for pmems in the Ring (that aren't None), $\mathbb{R} \geq 0$
- average_fit := total_fit / ring.num_filled
- best_pmem : the ring index for the pmem in the ring with the highest fitness value, ring.num_slots $\geq \mathbb{Z} \geq 0$
- best_fit : the value of the best fitness as found in the ring, $\mathbb{R} \geq 0$

14.4.2 Environment Variables

The output files with extension OUTPUT_FORMAT will be written to the current working directory.

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

run_output(ring):

- transition: generate an output files for the best conformer geometries.
- output: returns the average fitness value in the ring, the best fitness value in the ring, the energies of the best geometries, and the final ring object.
- exception: raise an error if the user doesn't have write permissions in the current working directory.

14.4.5 Local Functions

None

15 MIS of Geometry

The geometry module is used by the output module (Section 15), the ring module (Section 12), and the mol_input module (Section 7). It uses the external program Vetee to make a Parser object that is used by the ring. The Parser object can represent a few file formats: xyz, com, Gaussian log file (glog). The Parser object can also be initialized using a SMILES string, a cid (chemical identifier used by pubchem), or a molecule name. This module also converts zmatrices to xyz coordinates and vice versa.

15.1 Module

geometry

15.2 Uses

None

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
generate_parser	dict	Parser	NotImplementedError
zmatrix_to_xyz	str	list(list(str, $\mathbb{R}, \mathbb{R}, \mathbb{R}$))	None
generate_zmatrix	Parser, list(list($\mathbb{Z}$))	str	None

15.4 Semantics

15.4.1 State Variables

The Vetee Parser object has the following attributes:

- comments: str
- charge: charge of the molecule, $\mathbb{Z}$
- multip: multiplicity of the molecule, $\mathbb{N}$
- calc_type: str (optimization, single-point, etc.)
- coords: list(list(str, $\mathbb{R}, \mathbb{R}, \mathbb{R}$))
- gkeywords: dict (keys are Gaussian keywords, values are the arguments for the Gaussian keywords)

- fpath: filepath for the input file, str
- fname: filename for the input file, str

15.4.2 Environment Variables

None

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

generate_parser(mol_input_dict):

- transition: None
- output: Parser object.
- exception: NotImplementedError : struct_type is not covered by Vetee.

zmatrix_to_xyz(zmatrix):

- transition: None
- output: list(list[atom type (str), x-coord, y-coord, z-coord]) the xyz coordinates and the atomic types.
- exception: None

generate_zmatrix(parser, dihedrals):

- transition: None
- output: zmatrix (str) that is the combination of the parser.coords and the list(list(dihedral-angles)).
- exception: None

15.4.5 Local Functions

None

16 MIS of Energy

Using the psi4 program, this module is responsible for running energy calculations.

16.1 Module

energy

16.2 Uses

None

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
run_energy_calc	str, str, str, bool	$\mathbb{R}$	None
prep_psi4_geom	list(list[str, $\mathbb{R}, \mathbb{R}, \mathbb{R}$]), $\mathbb{Z}, \mathbb{N}$	str	None

16.4 Semantics

16.4.1 State Variables

- psi4_str : str representing the geometry specification in the format needed for a psi4 energy calculation.

16.4.2 Environment Variables

None

16.4.3 Assumptions

This module can be modified to support multiple programs depending on the user's preference for calculation software.

16.4.4 Access Routine Semantics

run_energy_calc(geom, method="scf", basis="aug-cc-pVTZ", restricted=False):

- transition: None
- output: energy ($\mathbb{R}$) of the geometry with the specified QCM and BS (which may be a restricted vs unrestricted calculation depending on the restricted bool).
- exception: None

prep_psi4_geom(coords, charge, multip):

- transition: None
- output: str of the psi4 geometry specification needed to perform calculations using a list of cartesian coordinates and atom names, the charge, and the multiplicity of the molecule.
- exception: None

16.4.5 Local Functions

None

17 MIS of RMSD

Uses the rmsd repository from github to calculate the root-mean-square deviation between all sets of conformer geometries in a pmem. Each pair is sent as xyz files to this module independently.

17.1 Module

rmsd

17.2 Uses

None

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
calc_rmsd	str, str	$\mathbb{R}$	None

17.4 Semantics

17.4.1 State Variables

- output := standard output from the rmsd module that is converted into a float and returned to the fitg module.

17.4.2 Environment Variables

Both files are in cartesian coordinates (can also have a pdb file extension).

- f1 := file name 1 for the first geometry.
- f2 := file name 2 for the second geometry.

17.4.3 Assumptions

The rmsd repository calculates a rotation matrix such that comparing molecules that have only undergone translation and rotation gives an rmsd of 0.

17.4.4 Access Routine Semantics

calc_rmsd(f1, f2):

- transition: open f1 and f2.
- output: calculated rmsd.
- exception: None

17.4.5 Local Functions

None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.