MMEA User's Guide

Lukas Turcani

January 20, 2017

Contents

1 Introduction

MMEA (MacroMolecular Evolutionary Algorithm) is a genetic algorithm (GA) for chemistry. It aims to be as general as possible, being suitable for use with any class of molecules. While it does not support all types of molecules out of the box, it is designed to be easily extended. This is made easier by being written in Python. For notes on how to extend MMEA to a new class of molecules or add GA operations, see the developer's guide.

2 Installing MMEA

3 Input Files

To run MMEA you need to use an input file. These contain all the details required to start a GA calculation. Among other things, this may include things like which fitness function you want to use, the number of generations MMEA should make and how big the population should be.

Using an example, the format and structure of input files is explained in section 3.1. A guide to finding out what GA operations and tools are available is in section 3.2.2. Examples of input files and accompanied explanations of what each line does are provided in section 3.3.

3.1 Format and structure.

The input file consists of a sequence of commands. Each command defines a variable or a function used by MMEA. If the command defines a function used by MMEA it must also define any parameters necessary to use the function. It does not have to define any default initialized parameters, though it may if desired. Commands can be written across multiple lines and terminate when the last parameter is defined. This means that

```
generational_select_func;
stochastic_sampling;
use_rank=True
```

and

```
generational_select_func; stochastic_sampling; use_rank=True
```

define the same command. Notice there is no semicolon at the end of a command, it is only used to separate the parameters.

Any lines consisting of only whitespace or beginning with "#" are ignored.

There are two kinds of command in the input file. The first kind has the form

```
keyword; function_name; param1=val1; param2=val2; param3=val3
```

For example,

```
generational_select_func; stochastic_sampling; use_rank=True
```

This means that the keyword is "generational_select_func", the function name is "stochastic_sampling", parameter 1's name is "use_rank" and its value is "True". As you can see, in this there is only 1 parameter name-value pair.

Commands with this format define functions used by MMEA. It is called the "function definition format". The function will correspond to the various GA operations, such as crossover, mutation and selection or to things like the fitness function or optimization function (which optimizes the geometry of molecules created by MMEA). The words which make up the command are separated by a semicolon. Recall that the last word of a command is not followed by a semicolon.

The keyword defines which GA operation the command describes. For example, in the command above the keyword "generational_select_func" signifies that the command defines which function is to be used for selecting members of the next generation. The keyword in the command

```
parent_select_func; crossover_roulette
```

is "parent_select_func", which signifies that the command defines the function used for selecting parents for crossover. A list of valid keywords is described in section 3.2.1.

The parameter "function_name" is the name of the function which is to carry out the role designated by the keyword. This must correspond to the name of a function defined within MMEA. For example, if the keyword is "generational_select_func" then "stochastic_sampling" or "roulette" are both valid values for "function_name", because the module "selection.py" within MMEA defines functions with these names. Indeed, any function defined within this module can be used as the selection function. Finding valid function names without looking at the source code is described in section 3.2.2.

Finally, in the command above "param1" and "val1" correspond to "use_rank" and "True", respectively. As mentioned before "stochastic_sampling" is the name of a function defined within the "selection.py" module. As is usual for Python functions, it takes arguments. Looking at "stochastic_sampling()" within "selection.py" it can be seen that its arguments are "cls" and "population" and its keyword arguments are "elitism" (defaults to "False"), "truncation" (defaults to "False"), "duplicates" (defaults to "False") and "use_rank" (defaults to "False").

In the case of selection functions the first two arguments ("cls" and "population") are dealt with automatically by MMEA. If the user wants to use one of the selection functions they have to provide values to any of the remaining arguments in the form "param1=val1". The order in which they are given in the input file does not matter. If the user wants to change the default values of any of the function's keyword arguments, they may add them to the command using the same notation. Keyword arguments otherwise do not have to be provided in the input script and will be default initialized to whatever is defined within the source code of MMEA. All of this holds true for other functions defined in the input file, not just selection functions. How a user can easily view what functions and parameters are available, as well as what they mean, is addressed in section 3.2.

When the user provides the value of a parameter, it must reflect its type. For example if a parameter wanted an a list of numbers then

```
param=[1,2,3,4]
```

if it wanted a string

```
param="this is a string"
```

or if it wanted a the name of a function or class defined within MMEA

```
param=some_mmea_function_name
```

In essence it must be provided as valid Python syntax.

All keywords should be defined in the input file once. The exception to this is "mutation_func". It is quite possible that during a GA run the user's wishes to use multiple mutation operations. As a result it is perfectly reasonable to include multiple commands with the keyword "mutation_func". When this is done, the user must also include the line

```
mutation_weights=[0.1, 0.45,0.45]$
```

This line tells MMEA that when a mutation is to be performed it should select the first mutation function defined in the input file with a 0.1 probability and the second and third mutation functions defined in the input file with a 0.45 probability. The probabilities can of course be modified by the user. The only requirements are that the probabilities sum to 1 and that the number of elements in the array and the number of mutation functions defined in the input file is the same.

There is a second kind of non-comment line in the input file of MMEA. These lines have the form

```
keyword=val$
```

As you can see, these are much simpler. They define basic constants relevant to the GA. As a result, this is called the "constant definition format". This includes things like population size, number of crossovers carried out during each generation and so on. It is merely the keyword, a "=" and the value. No spaces. Valid keywords for these lines are discussed in section 3.2.

3.2 Valid input values.

In order to prevent the need to continuously update this documentation when a new function is added to MMEA, only valid keywords are described here. However, this does not mean a user has to go into the source code to find out what functions they have at their disposal. This section also describes how MMEA can be made to output valid function names and parameters for each keyword.

3.2.1 List and explanation of valid keywords.

Here is a list of valid keywords, the type of line formatting the keyword requires and which arguments do not need to be provided to the functions. Recall in section 3.1 the selection function did not require the "population" argument to be provided explictily in the input file. Other functions, such as crossover functions do not require other parameters to be provided. The format here is:

keyword - *line format* - **excluded_param1, excluded_param2** - Explantion of keyword.

The keywords are:

init_func - *function definition format* - **ga_tools** - The function used to create the initial population.

generational_select_func - *function definition format* - **population** - The function used to select members of the next generation.

parent_select_func - *function definition format* - **population** - The function used to select parents for crossover.

mutant_select_func - *function definition format* - **population** - The function used to select individuals for mutation.

crossover_func - *function definition format* - **marco_mol1, macro_mol2** - The function which carries out the crossover operation.

mutation_func - *function definition format* - **macro_mol** - The function which carries out the mutation operation.

opt_func - *function definition format* - **macro_mol** - The function which optimizes the geometry of the molecules.

fitness_func - *function definition format* - **macro_mol, population** - The function which calculates fitness.

num_generations - *constant definition format* - The number of generations which MMEA will create.

num_mutations - *constant definition format* - The number of mutations MMEA will carry out each generation.

num_crosses - *constant definition format* - The number of crossover operations MMEA will carry out each generation.

pop_size - *constant definition format* - The size of the population.

mutation_weights - *constant definition format* - The probability with which each mutation function defined in the input file will be used when a mutation operation is performed. Note that the form of this constant is an array. For example

```
mutation_weights=[0.1, 0.45,0.45]$
```

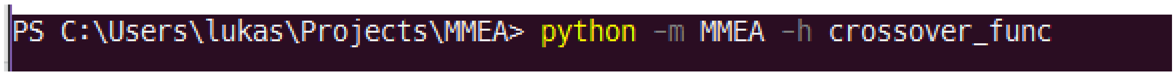
3.2.2 Finding available functions and their arguments.

In order to find out what functions MMEA offers for a given keyword, using a console go into the directory containing MMEA.

Run the command

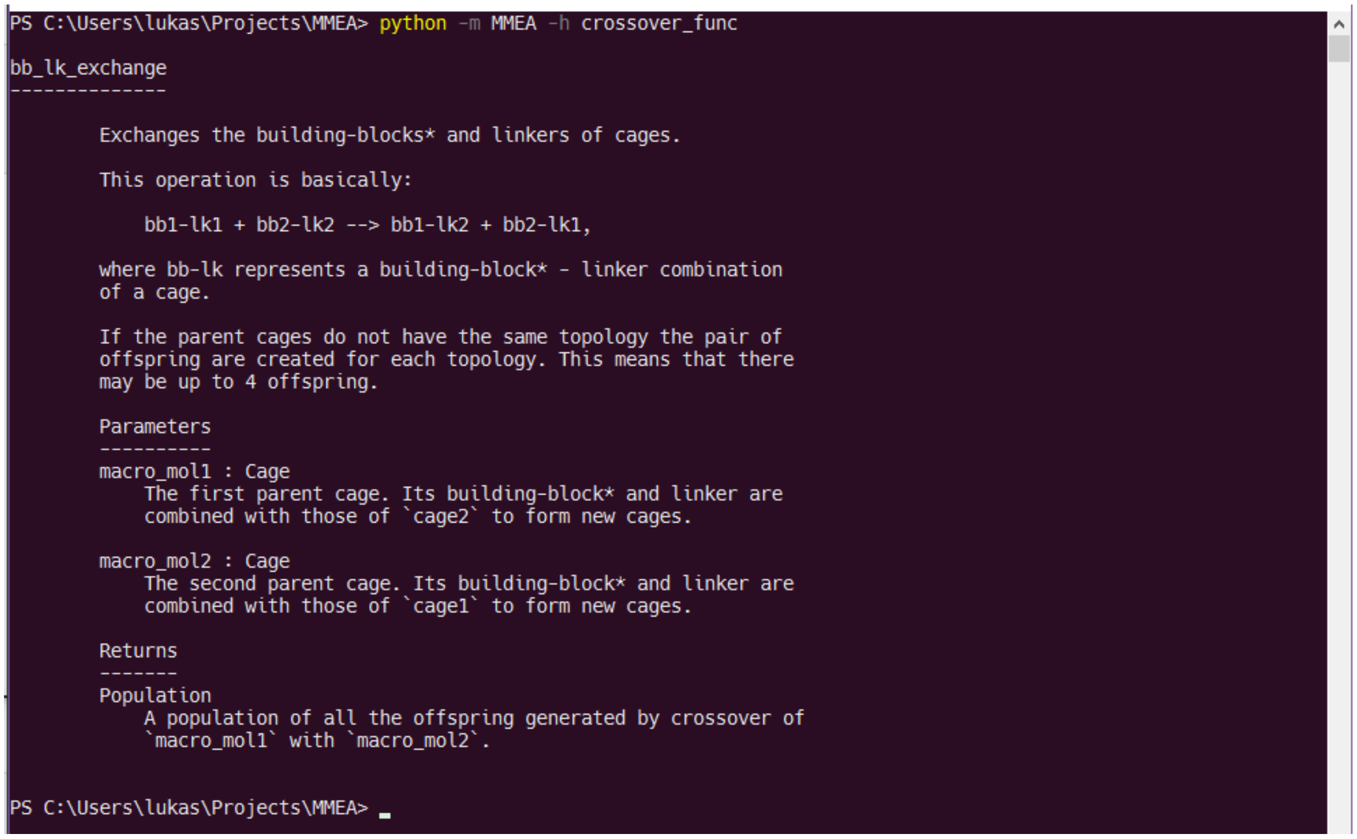
```
python -m MMEA -h keyword
```

where keyword is one of the valid function definition keywords defined in section 3.2.1.



```
PS C:\Users\lukas\Projects\MMEA> python -m MMEA -h crossover_func
```

MMEA will then output a list of valid functions, their description, and a list of their parameters.



```
PS C:\Users\lukas\Projects\MMEA> python -m MMEA -h crossover_func

bb_lk_exchange
--------------

        Exchanges the building-blocks* and linkers of cages.

        This operation is basically:

            bb1-lk1 + bb2-lk2 --> bb1-lk2 + bb2-lk1,

        where bb-lk represents a building-block* - linker combination
        of a cage.

        If the parent cages do not have the same topology the pair of
        offspring are created for each topology. This means that there
        may be up to 4 offspring.

        Parameters
        ----------
        macro_mol1 : Cage
            The first parent cage. Its building-block* and linker are
            combined with those of `cage2` to form new cages.

        macro_mol2 : Cage
            The second parent cage. Its building-block* and linker are
            combined with those of `cage1` to form new cages.

        Returns
        -------
        Population
            A population of all the offspring generated by crossover of
            `macro_mol1` with `macro_mol2`.


PS C:\Users\lukas\Projects\MMEA> _
```

In this example, the keyword "crossover_func" was used and the only available function was "bb_lk_exchange" Other keywords will have more options available (and more may be added to this one). As you can see the only parameters the function takes are macro_mol1 and macro_mol2. As these are handled automatically by MMEA (3.2.1) no parameters should be provided in the input file. A valid line would look like

```
crossover_func; bb_lk_exchange$
```

3.3 Examples

Here are example input files followed by a line by line explanation of what they're doing.

3.4 Example 1

```
#################################################################################
# Population initialization function.
#################################################################################

init_func; init_random_cages;
bb_db="/home/username/aldehydes_3f";
lk_db="/home/username/amines_2f";
```

```
topologies=[FourPlusSix, EightPlusTwelve]$

#################################################################################
# Selection function for selecting the next generation.
#################################################################################

generational_select_func; stochastic_sampling; use_rank=True$

#################################################################################
# Selection function for selecting parents.
#################################################################################

parent_select_func; crossover_roulette$

#################################################################################
# Selection function for selecting molecules for mutation.
#################################################################################

mutant_select_func; stochastic_sampling; duplicates=True$

#################################################################################
# Crossover function.
#################################################################################

crossover_func; bb_lk_exchange$

#################################################################################
# Mutation function 1.
#################################################################################

mutation_func; similar_bb;
database="/home/username/aldehydes_3f"$

#################################################################################
# Mutation function 2.
#################################################################################

mutation_func; similar_lk;
database="/home/username/amines_2f"$

#################################################################################
# When carrying mutations, chance that a given mutation function will be used.
#################################################################################

mutation_weights=[1/2,1/2]$

#################################################################################
```

```
55 # Optimization function.
56 ##################################################################################
57
58 opt_func; macromodel_cage_opt;
59 macromodel_path="/home/username/program_files/schrodinger2016-3"$
60
61 ##################################################################################
62 # Fitness function.
63 ##################################################################################
64
65 fitness_func; cage; target_cavity=8; coeffs=[1,1,5,0,0];
66 energy_params={"key":("macromodel", 16, "/home/username/schrodinger2016-3")}$
67
68 ##################################################################################
69 # Number of generations to create.
70 ##################################################################################
71
72 num_generations=10$
73
74 ##################################################################################
75 # Number of mutation operations to perform each generation.
76 ##################################################################################
77
78 num_mutations=10$
79
80 ##################################################################################
81 # Number of crossover operations to perform each generation.
82 ##################################################################################
83
84 num_crossovers=15$
85
86 ##################################################################################
87 # Size of the population.
88 ##################################################################################
89
90 pop_size=20$
```

The first command runs from line 5 to 8. It has the keyword "init_func", which means that it defines the initialization function for the GA population. The initialization function is called "init_random_cages". This function creates a population consisting of randomly assembled "cage" molecules. The function takes the parameter "bb_db" which receives a string. The string is the path of a directory which holds monomers used for constructing the cages. The parameter "lk_db" has the same role. The parameter "topologies" requires an array or list. The list contains the names of cage topologies defined within MMEA.

The second command is on line 14. The keyword "generational_select_func" indicates that the command defines the selection function used for selecting the next generation.

4 Running MMEA

4.1 From the command line.

5 Output

6 MMEA as a Library