

**Genetic programming types comparison**

**Matteo Gianvenuti**

Summary

[Project description 3](#_Toc190371128)

[Changes made to the reference code 3](#_Toc190371129)

[data\_loader.py 3](#_Toc190371130)

[utils.py 3](#_Toc190371131)

[gp\_types.py 3](#_Toc190371132)

[user\_interface.py 3](#_Toc190371133)

[user\_interface\_charts.py 3](#_Toc190371134)

[Representation complexity analysis 3](#_Toc190371135)

[Complexity of *extraction\_tree* 3](#_Toc190371136)

[Complexity of *get\_modules\_tree* 4](#_Toc190371137)

[Complexity of *extraction\_list* 4](#_Toc190371138)

[Complexity of *get\_modules\_list* 5](#_Toc190371139)

[Final complexity comparison 5](#_Toc190371140)

[Comparison 5](#_Toc190371141)

# Project description

The.

# Changes made to the reference code

General changes:

* Switch to English.
* Use of Python best practices.
* Switch to f strings, they are more efficient because avoid concatenation and calls to the “str” function.

## data\_loader.py

The original file was called “utils\_import\_data.py”. I removed the function “shuffle\_in\_union”. It is sufficient to use the shuffle=True parameter in “train\_test\_split” function already available from scikit-learn.

## utils.py

The original file was called “utils\_functions.py”.

## gp\_types.py

The original file was called “GPmodular.py”.

## user\_interface.py

## user\_interface\_charts.py

# Representation complexity analysis

The “representation functions” are in *utils.py*. For clarity, I added the postfix *\_tree* to all the functions from the Cella’s reference code that works on a tree representation and *\_list* to all the functions from the Stefano’s reference code that works on a list representation (interested functions: extraction and get\_modules). For the following analysis, “n” is the general length of a string (individual).

## Complexity of *extraction\_tree*

The “replace” function have a complexity of O(n).

*regex\_depth1* complexity: *(?:add|sub|neg|mul|div|execTree\d+)* is a direct comparison so it is O(1), the part *\((...)\)* is a series of alternatives:

* -?\d+ search for a positive/negative number, worst case O(n)
* [A-Za-z0-9\_]+ search one or more objects in the specified ranges, worst case O(n)
* \([^()]+\) search for something between parenthesis but not parenthesis, worst case o(n)
* -?\d+,-?\d+ search for a pair of integer, worst case O(n)
* [-A-Za-z0-9\_]+,-?\d+ search for a pair “string” in the range and a number, worst case O(n)
* [-A-Za-z0-9\_]+,[A-Za-z0-9\_]+ search for a pair of “strings” in the range, worst case O(n)

All the other regex are combinations of these or similar objects like “(?:-?\d+|ARG\d+|[A-Za-z0-9\_]+)”, “(?:,-?\d+|,ARG\d+|,[A-Za-z0-9\_]+){0,3}” this is O(3n) but it is always O(n), etc. So, all these have a complexity of O(n). As consequence the re.findall(str\_a, str\_b) has a complexity O(m\*n) where m is the number of matches but it is always O(n). We know that O(n) + O(n) + … + O(n) = O(n). For the last part of the function is better to see the code with comments.

A computer screen with text on it

AI-generated content may be incorrect.

The original code was the one with worst case complexity O(n2), with my optimization the final complexity of this function is O(n).

## Complexity of *get\_modules\_tree*

This function is composed by a for loop (on the population of size p) with a call to *extraction\_*tree and two for loop (one for module with size m1 and m2) nested at the same level, so the complexity is O(p)\*(O(n)+O(m1)+O(m2)) = O(p\*n). It could be considered as O(n) with n → ∞ or O(p) with p → ∞ and a limitation on the size of the individuals.

## Complexity of *extraction\_list*

The original complexity of this function wrote by Stefano was O(n) but there was some overhead due to the support structures and the entire tree traversal making it a lot slower than the optimized tree version. The function was also incorrect as I said before.

Out of this there is another problem, I noticed the parsing of the tree is wrong, terminals are considered as internal nodes, this build an incorrect tree structure. In fact, commas separate children of the same parent, but the code nests subsequent nodes under the previous terminal.

## Complexity of *get\_modules\_list*

The structure of this function is similar to the tree version, but it needs to do some more O(1) checks because the *exctraction\_list* does not return the individual already separated in submodules. The external loop is always O(p).

## Final complexity comparison

The regex version is better, it is intuitive because it avoids explicit tree traversal and directly identifies valid subtrees of depth 1 or 2 with a direct match, reducing overhead.

# Results comparison