

**Genetic programming types comparison**

**Matteo Gianvenuti**

Summary

[Project description 3](#_Toc190277002)

[Changes made to the reference code 3](#_Toc190277003)

[data\_loader.py 3](#_Toc190277004)

[utils.py 3](#_Toc190277005)

[gp\_types.py 3](#_Toc190277006)

[user\_interface.py 3](#_Toc190277007)

[user\_interface\_charts.py 3](#_Toc190277008)

[Representation complexity analysis 3](#_Toc190277009)

[Comparsion 3](#_Toc190277010)

# Project description

The.

# Changes made to the reference code

The.

## data\_loader.py

## utils.py

## gp\_types.py

## user\_interface.py

## user\_interface\_charts.py

# Representation complexity analysis

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, get\_modules, get\_modules\_individual and depth). For the following analysis, “n” is the general length of a string.

## 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 O(n\*\*2), 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 with a call to *extraction\_*tree and two for loop nested at the same level, so its complexity is basically O(n\*O(3\*n)) = O(n\*\*2).

## Complexity of *get\_modules\_individual\_tree*

This function call the *extraction\_tree* to get the modules and after join them in one list so it is O(n).

# Comparison