Análise Evolutiva da Interconexão das Linguagens de Programação

1st Rafael Augusto Campos Moreira
DIGD
Centro Federal de Educação Tecnológica de Minas Gerais
Divinópolis-MG, Brasil
Camposrafa806@gmail.com

***Abstract*—Este artigo possui o objetivo de realizar uma investigação evolutiva acerca da interconexão das diversas linguagens de programação existentes. Para que isso fosse possível, utilizou a estrutura de dados grafo, a fim de compreender como a progressão científica relacionada a conceitos linguísticos, os quais são utilizados no desenvolvimento computacional, se dá ao longo do tempo, tendo em vista que nem sempre se tem um grande conhecimento das linguagens que mais foram utilizadas juntas em um determinado momento da história, seja por semelhança, por harmonia colaborativa ou por outro fator.**

***Index Terms*—Linguagens, Programação, Interconexão, Evolução**

I. Introdução

Com o advento da era moderna, é evidente que os sistemas computacionais se tornaram cada vez mais presentes nas rotinas diárias de grande parte da população, auxiliando nas mais diversas situações, como automatização de processos, busca de soluções de problemas e uso de tecnologias. Para que tais sistemas possam fazer isso com excelência, instruções referentes a procedimentos e comandos, detalhadas por meio da comunicação homem-máquina (HMI), são imprescindíveis, tendo em vista que o grau de eficiência de execução de um algoritmo é diretamente proporcional à capacidade de eloquência da linguagem usada para a transmissão de tais instruções.

Nesse sentido, as linguagens de programação surgiram com o intuito de fornecer instruções referentes a procedimentos e comandos para os sistemas computacionais, sendo classificadas como ferramentas de grande importância para o constante advento tecnológico.

Por definição, sistemas computacionais são conjuntos de componentes eletrônicos (hardware) e programas (software) que trabalham juntos para realizar uma tarefa específica (Andrew S. Tanenbaum e Herbert Bos). Para que esses programas sejam criados, ferramentas conhecidas como linguagens de programação são utilizadas, as quais se definem como sistemas formais que especificam instruções e comandos para um computador (Alfred V. Aho, Monica S. Lam, Ravi Sethi e Jeffrey D. Ullman).

Assim, afirma-se que desconsiderar a importância das linguagens de programação como catalisadores de execução de tarefas e automatização de softwares é deixar de perceber a realidade atual. Logo, uma investigação evolutiva acerca da interconexão dessas linguagens com o decorrer do tempo pode ser compreendida como um conjunto de métodos que visa a progressão científica relacionada a conceitos linguísticos, os quais são utilizados no desenvolvimento computacional ao longo do tempo, tendo em vista que nem sempre apenas uma só linguagem é capaz de suprir as demandas de uma determinada tecnologia.

As análises evolutivas da interconexão das linguagens de programação se concentram em descobrir o porquê da combinação de determinadas linguagens terem sido escolhidas como a mais apropriada e eficiente para o processamento de uma determinada tecnologia em um momento da história, e qual o impacto disso nos futuros progressos tecnológicos envolvendo a comunidade em geral. Desse modo, o estudo evolutivo das diversas linguagens de programação existentes, com enfoque em suas interconexões, pode ser utilizado como ferramenta de auxílio no processo de desenvolvimento científico e aprendizagem computacional.

Em se tratando de evidências concretas sobre desenvolvimentos de softwares e tecnologias, têm-se o GitHub, uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git. Ele permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribuam em projetos privados e Open Source de qualquer lugar do mundo. Nesse sentido, uma base de dados de tal plataforma, contendo informações sobre as linguagens utilizadas em seus respectivos repositórios (locais que se armazenam o software a ser desenvolvido), nos anos de 2010, 2015 e 2020, pode ser usada para o entendimento da relação do desenvolvimento da tecnologia com as ferramentas utilizadas (linguagens).

Portanto, esse artigo visa a pesquisa e a tomada de conclusões pertencentes ao uso de determinadas linguagens interconectadas para o progresso tecnológico temporal.

II. Trabalhos Correlatos

É evidente que o GitHub permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma possam publicar seus trabalhos, dúvidas e contribuições relacionadas à área de desenvolvimento computacional. Em virtude disso, tal plataforma se torna uma imensa base de dados, contendo informações sobre repositóios, desenvolvedores, linguagens utilizadas e tendências tecnológicas em alta.

Em se tratando diretamente de uma mineração de dados do GitHub, cita-se Cláudio Oliveira, Rogério Tostes e Tassio Ferenzini Martins Sirqueira (2021) como grandes contribuidores, uma vez que desenvolveram o *GIT Viewer*, uma plataforma para análise de dados do GitHub.

Além disso, Lais M. A. Rocha, Thiago Henrique P. Silva e Mirella M. Moro (2016) exploraram as diversas contribuições existentes entre repositórios do GitHub, buscando traçar uma correlação entre os desenvolvedores com o intuito de estabelecer espécies de pontes entre a rede social, a partir das linguagens de programação escolhidas. Por fim, na caracterização da qualidade de código, Ray et al. (2014) estuda os tipos e os usos de LPs e como se relacionam à qualidade de software.

Dessa forma, todos os artigo citados anteriormente contribuíram para se ter a visão da busca de novas soluções com o uso de dados do GitHub. Também, fomentaram o conhecimento a respeito das linguagens de programação e as possíveis conexões realizadas entre os repositórios.

III. Metodologia

Para se construir a análise evolutiva da interconexão das linguagens de programação, utilizando como base de dados os repositórios do GitHub de 2010, 2015 e 2020, foi-se necessário extrair um grande conjuntos de dados da plataforma de datasets *Kaggle*. Originalmente, a base de dados, intitulada como *repositories.csv*, possui dezesseis anos de registros sobre dois mil novecentos e cinquenta e cinco repositórios. Todo repositório é classificado em vinte e uma características diferentes: *repo id, owner username, repo name, description, created at, pushed at, size, stargazers count, has projects, has wiki, has pages, forks count, open issues count, license, is template, topics, watching, contributors count, commits count, languages e readme*.

Dessas características, *created at* e *languagens* foram classificadas como as mais importantes, uma vez que o objetivo do trabalho em questão é traçar uma análise temporal da interconexão das linguagens de programação. Para que essa importância fosse definivamente consolidada, uma raspagem nos dados se tornou necessária, a fim de descartar as características menos significativas. Dito isso, a biblioteca *pandas* da linguagem de programação Python foi escolhida como a mais adequada para esse processo. O motivo de tal escolha foi porque, segundo Wes McKinney (2017), essa biblioteca se classifica como uma das mais amplas para análise de dados, oferecendo estruturas como *Dataframe* e *Series*.

Por definição, um *DataFrame* é um recipiente para dados de séries temporais. Em outras palavras, *pandas DataFrame* é uma estrutura de dados bidimensional com colunas potencialmente diferentes de tipos de dados, semelhante a uma tabela de banco de dados ou uma planilha do Excel. Já uma *Serie* é uma estrutura unidimensional que pode armazenar qualquer tipo de dado, podendo ser vistas como uma coluna em uma planilha do Excel ou como uma única coluna em um *DataFrame* (W. McKinney, 2017).

Dispondo de tais recursos, removeu-se as seguintes características: *repo id, pushed at, size, has projects, has wik, has pages, license, is template, topics, watching, commits count e readme*. É importante salientar que o motivo da conservação das informações restantes (*owner username, repo name, description, created at, stargazers count, forks count, open issues count, contributors count e languages*) é que tais informações podem ser utilizadas como atributos de interconexões entre os repositórios em trabalhos futuros, com o intuito de aprofundar a investigação e a contribuição científica.

Feito tais remoções, observou-se que mais tratamentos precisavam ser feitos na base de dados, tendo em vista que nem todos os repositórios possuíam duas ou mais linguagens de programação utilizadas. Esse fato foi de extrema importância porque, para se construir uma interconexão, é necessário dois ou mais elementos. Assim, apenas foram conservados os repositórios que possuem duas ou mais linguagens associadas.

Por fim, uma última manipulação se deu no tópico *created at*. Originalmente, tal tópico informa o dia, mês, ano e horário de criação daquele repositório e, para o desenvolvimento dess trabalho em questão, o ponto mais interessante é apenas o ano de criação, uma vez que a análise evolutiva é quinquenal. Possuindo todos os tratamentos concluídos, partiu-se para a parte da criação da estrutura escolhida para melhor armazenar os dados e elaborar a interconexão. Nesse aspecto, a estrutura escolhida foi grafo.

Um grafo é uma estrutura matemática que consiste em um conjunto de vértices (ou nós) e um conjunto de arestas que conectam pares de vértices (T. H. Cormen, 2009). A análise e modelagem de sistemas complexos encontram na teoria dos grafos um poderoso instrumento, onde vértices e arestas se entrelaçam para decifrar relações e padrões subjacentes. Em um mundo cada vez mais interconectado, os grafos emergem como uma ferramenta essencial para compreender a tessitura intricada da informação e dinâmicas interdependentes.

Para se construir a análise evolutiva, escolheu-se três anos dos dezesseis disponíveis (2010 - 2015 - 2020), levando em consideração número de repositórios, linguagens utilizadas e padronização temporal (de cinco em cinco anos).

Após essa decisão, iniciou-se a construção dos três grafos da seguinte forma: o vértice (elemento que será conectado, entidade) são todas as linguagens utilizadas naquele determinado ano e as aretas (relação ou conexão entre entidades) são a conexão entre duas linguagens de programação. Além disso, após construída essa estrutura, como o objetivo desse trabalho é realizar um estudo evolutivo da interconexão das linguagens, seu ápice se torna evidente quando se descobre quais as ligações mais fortemente conectadas em uma determinada época, ou seja, quais nós possem as arestas mais ponderadas. Por esse motivo, utilizou-se o algoritmo de Kruscal (Árvore Geradora Máxima), cujo objetivo é encontrar a árvore geradora de peso máximo (arestas de maior carga) em um grafo conexo e não direcionado (as arestas não têm uma direção específica), sendo classificada como um subgrafo que contém todos os vértices do grafo original (Kruskal, J. B., 1956).

A seguir, segue-se o pseudocódigo da criação do grafo em questão:

Algorithm 1: Adição de Vértices e Arestas

Data: Repositories_Ano (2010 - 2015 - 2020)
Result: Graph

```
for linha ← Repositories_Ano do
    languageslist = linguagens ← linha
     aux ←languages[0] cont ←0
    for elemento ←languages list do
        aux ←languages list[cont]
        cont+ = 1
        for lang ←languages list do
            if aux ≠lang then
                if aux ≠Dict Lang then
                    DicitLing[aux] ←lang : 1
                else
                    if lang ←Dict Lang[aux] then
                        DictLang[aux][lang]+ =1
                    else
                        DictLang[aux][lang]+ =1
for No1, listadjacent ←Dict Lang do
    for No2, Peso ←list adjacent do
        Graph ←No 1, No 2, Peso
```

É importante salientar que, para os anos de 2015 e 2020, adaptações foram feitas no algoritmo de Kruskal, tendo em vista que, nesses anos, em virtude do grande número de repositórios e linguagens, a Geradora Máxima possuía arestas com pesos despresíveis em relação as arestas extremamente ponderadas. Por esse motivo, tais arestas foram desconsideradas. Precisamente, para o ano de 2015, qualquer aresta na Geradora Máxima com peso menor ou igual à 3 foi desconsiderado. Para o ano de 2020, esse valor foi alterado para 5. O porquê de tais valores está relacionado à grande quantidade de arestas desconsideráveis nesse intervalo.

Algorithm 2: Criação Gerado Máxima

Data: Graph
Result: Maximo Graph

```
Maximo ←Graph.Maximo()
Adaptação ←Valor
for Aresta ←Maximo.Arestas() do
    if Aresta > Adaptação then
        MaximoGraph ←Maximo
```

IV. Resultados

Como resultado, obteve-se 3 grafos, cada um com seu número de nós e arestas específicos. Após isso, para se descobrir as linguagens de programação que mais se interconectavam, aplicou-se o algoritmo de algoritmo de Kruskal (Árvore Geradora Máxima), conforme citado anteriormente. A seguir encontram-se as geradoras máximas. Por motivos de visualização, os grafos completos estão em anexo no final do artigo, seguindo a ordem temporal (2010 - 2015 - 2020).

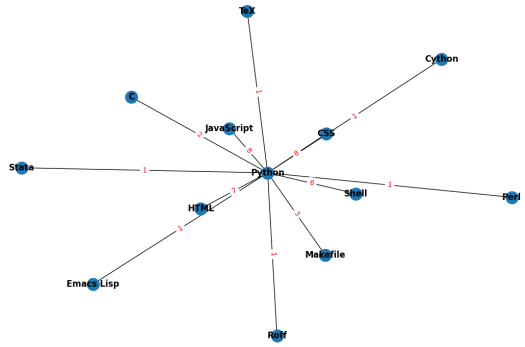


Fig. 1. Geradora Máxima - 2010.

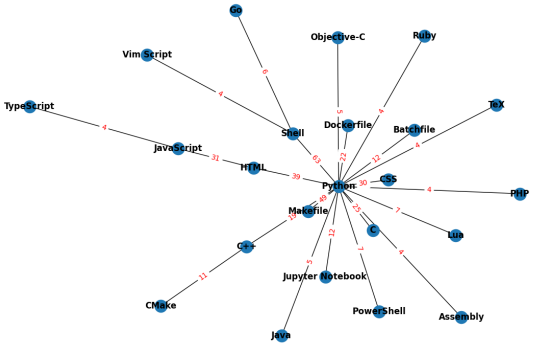


Fig. 2. Geradora Máxima - 2015.

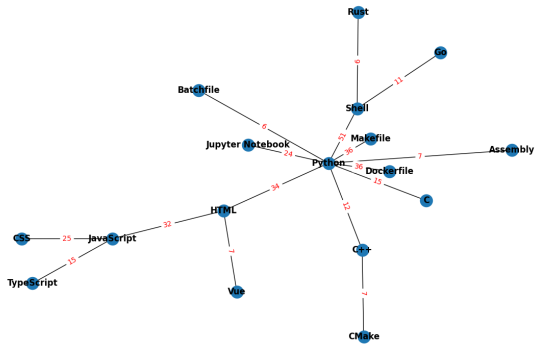


Fig. 3. Geradora Máxima - 2020.

Interpretando os grafos e as geradoras máximas obtidas, têm-se a tabela com as linguagens mais interconectadas em

um determinado ano:

Linguagens	Número de Conexões	Ano
Python e CSS	7	2010
Python e JavaScript	7	2010
Python e HTML	6	2010
Python e Shell	5	2010
Python e Shell	45	2015
Python e Makefile	35	2015
Python e HTML	29	2015
HTML e JavaScript	22	2015
Python e Shell	40	2020
Python e HTML	29	2020
Python e Makefile	28	2020
Python e JavaScript - Dockerfile	27	2020

Por fim, têm-se a seguinte análise evolutiva:

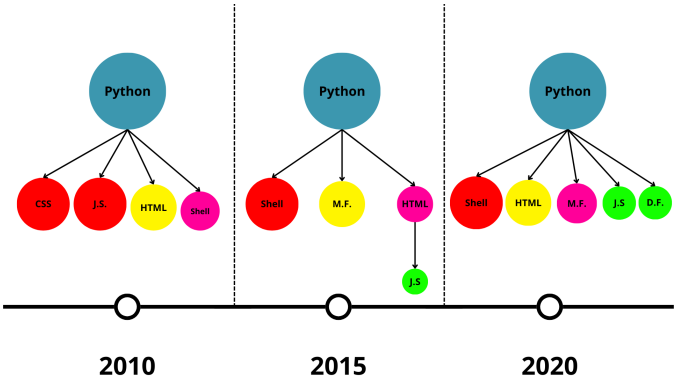


Fig. 4. Análise Evolutiva.

Tal imagem possui uma associação entre cores, tamanhos e números de ponderamento. Quanto maior a linguagem, maior é o número de vezes que ela apareceu naquele tempo. A setas indicam as interconexões, e as cores valores fixos expressos na tabela.

V. Conclusão

Em síntese, este artigo investigou a interconexão das linguagens de programação dado um momento histórico. Tendo as informações e dados apontados na análise, afirma-se que Python é a linguagem dominante, apresentando diversas conexões independente do período em que se encontra. Tal fato está relacionado principalmente à sua versalidade, sintaxe clara, comunidade ativa e diversidade de bibliotecas. Em se tratando das interconexões, não restam dúvidas que, majoritariamente, a interconexão mais forte se dá entre as linguagens Python e Shell. Isso pode ser explicado por meio da ampla capacidade que Python tem de interagir com Tecnologias Web, como Shell, HTML, CSS e JavaScript.

Por fim, essas descobertas não apenas contribuem para a compreensão atual de como as linguagens de programação podem se interconectar em um mesmo projeto, mas também oferecem implicações significativas para a compreensão linguistica. Apesar de Sheel e Python possuírem diferenças semânticas significativas, tais linguagens se mostram como sinérgicas (mesmo objetivo). No entanto, é importante reconhecer as limitações deste estudo, como os poucos momentos históricos abordados e a irrelevância de certas características como ponderamento da interconexão. Como sugestão para futuras pesquisas, recomenda-se abordar os tópicos citados anteriormente, a fim de se contruir uma análise mais ampla e estatísticamente válida.

Em última análise, este trabalho reforça a importância contínua das linguagens de programação no desenvolvimento tecnológico, proporcionando uma base sólida para investigações futuras e avanços científicos nessa área.

References

[1] S. Arboricola, ”Github Bipartite graph dataset”, Kaggle, 2023. [Online]. Disponível: https://www.kaggle.com/datasets/aditijuneja/github-bipartite-graph-datasetdevelopersrepos

[2] A. S. Tanenbaum e H. Bos, ”Modern Operating Systems” (Vol. 4), 2014

[3] A. V. Aho, M. S. Lam, R. Sethi e J. D. Ullman, ”Compilers: Principles, Techniques, and Tools” (Vol. 2), 2006

[4] C. Oliveira, R. Tostes, e T. F. M. Sirqueira, “GIT Viewer: Uma plataforma para análise de dados do GitHub”, Centro Universitário Academia – UniAcademia, Brazil, 2021.

[5] L. M. A. Rocha, T. H. P. Silva, M. M. Moro, Análise da Contribuição para Código entre Repositórios do GitHub, UFMG, Brazil, 2016.

[6] Ray et al, B., A large scale study of programming languages and code quality ingithub. InSIGSOFT FSE, pages 155–165, 2014

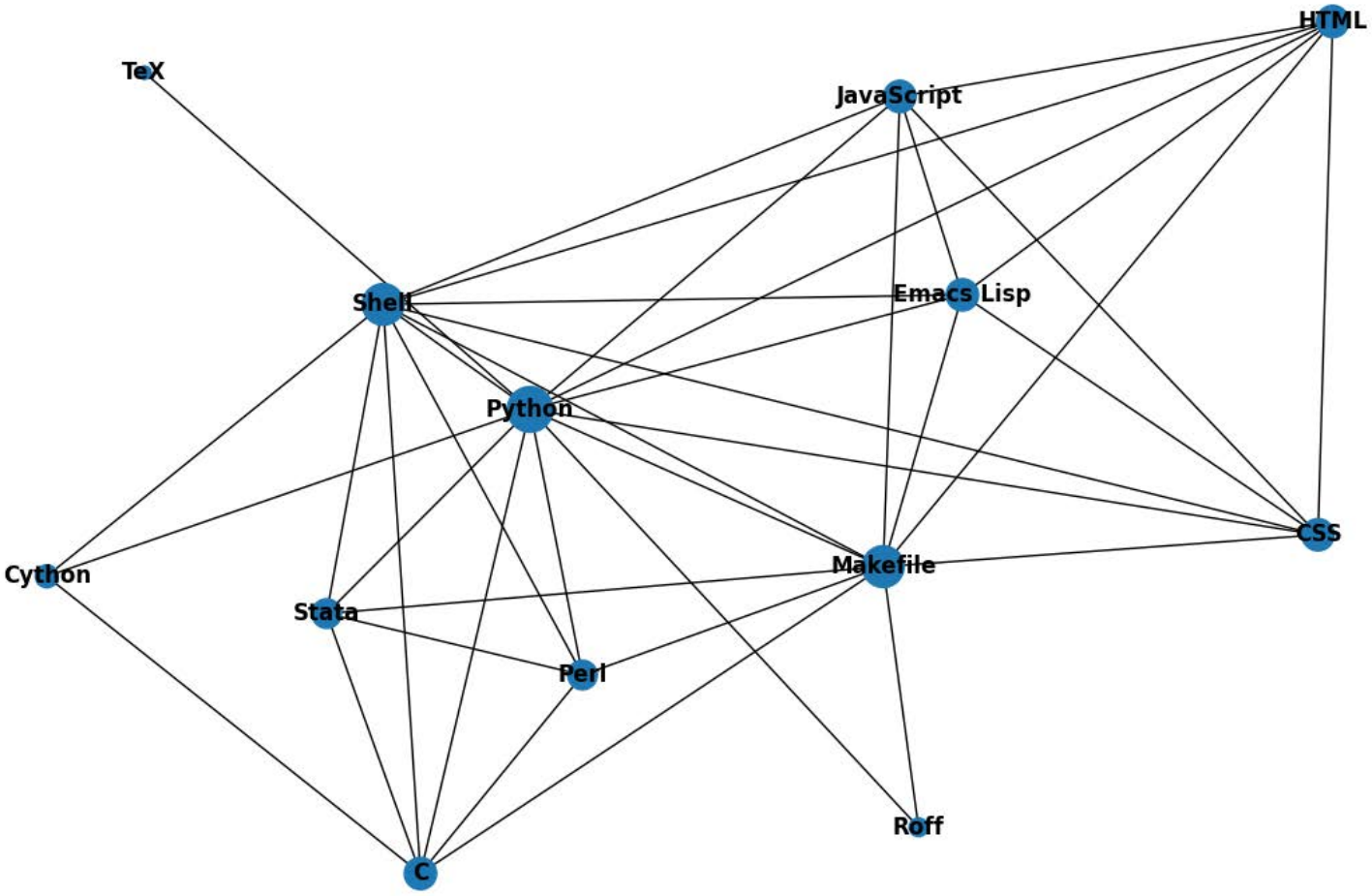
[7] W. McKinney, “Python for Data Analysis”, O’Reilly Media, Estados Unidos, 2017.

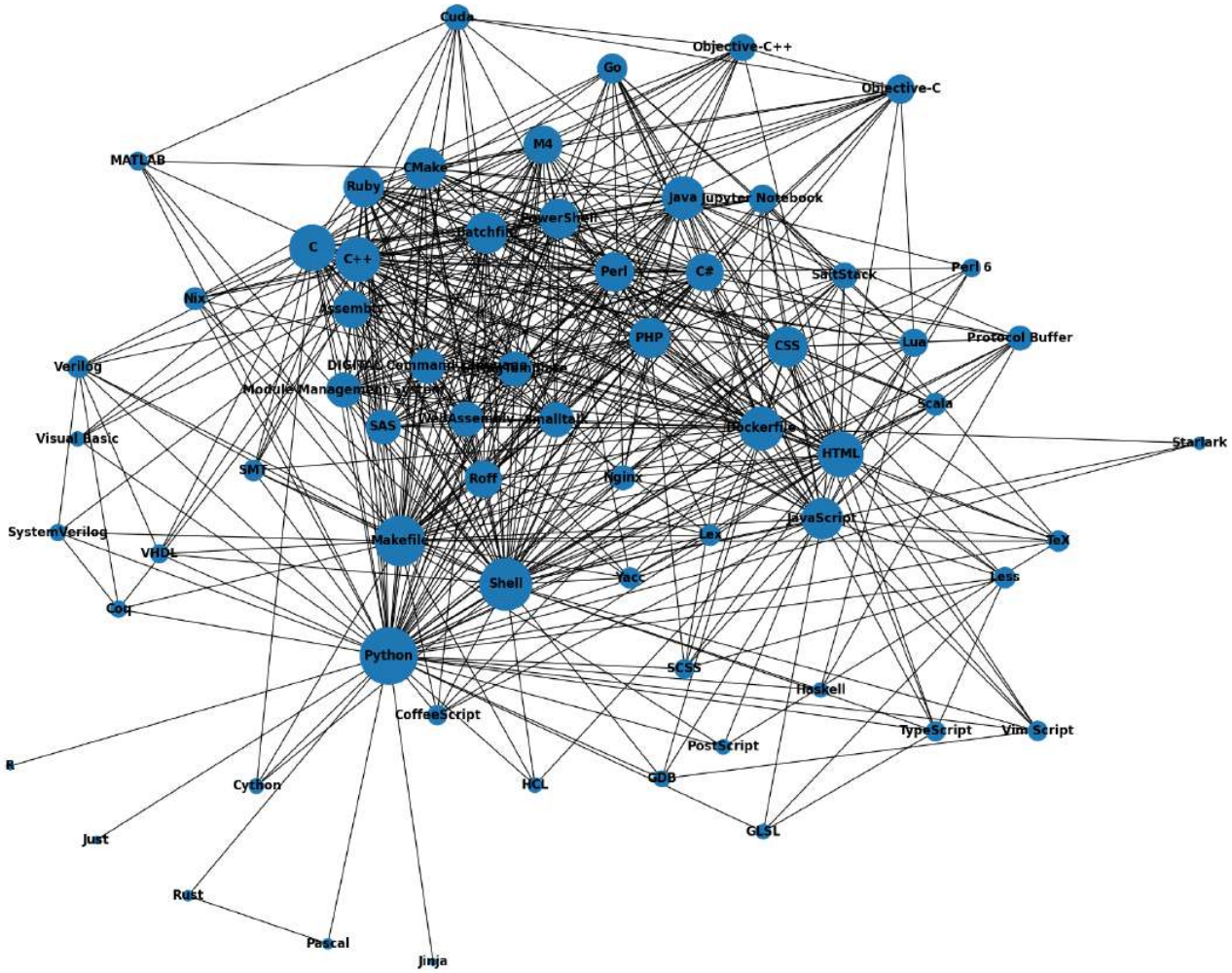
[8] T. H. Cormen, “Introduction to Algorithms”, MIT, Estados Unidos, 2012.

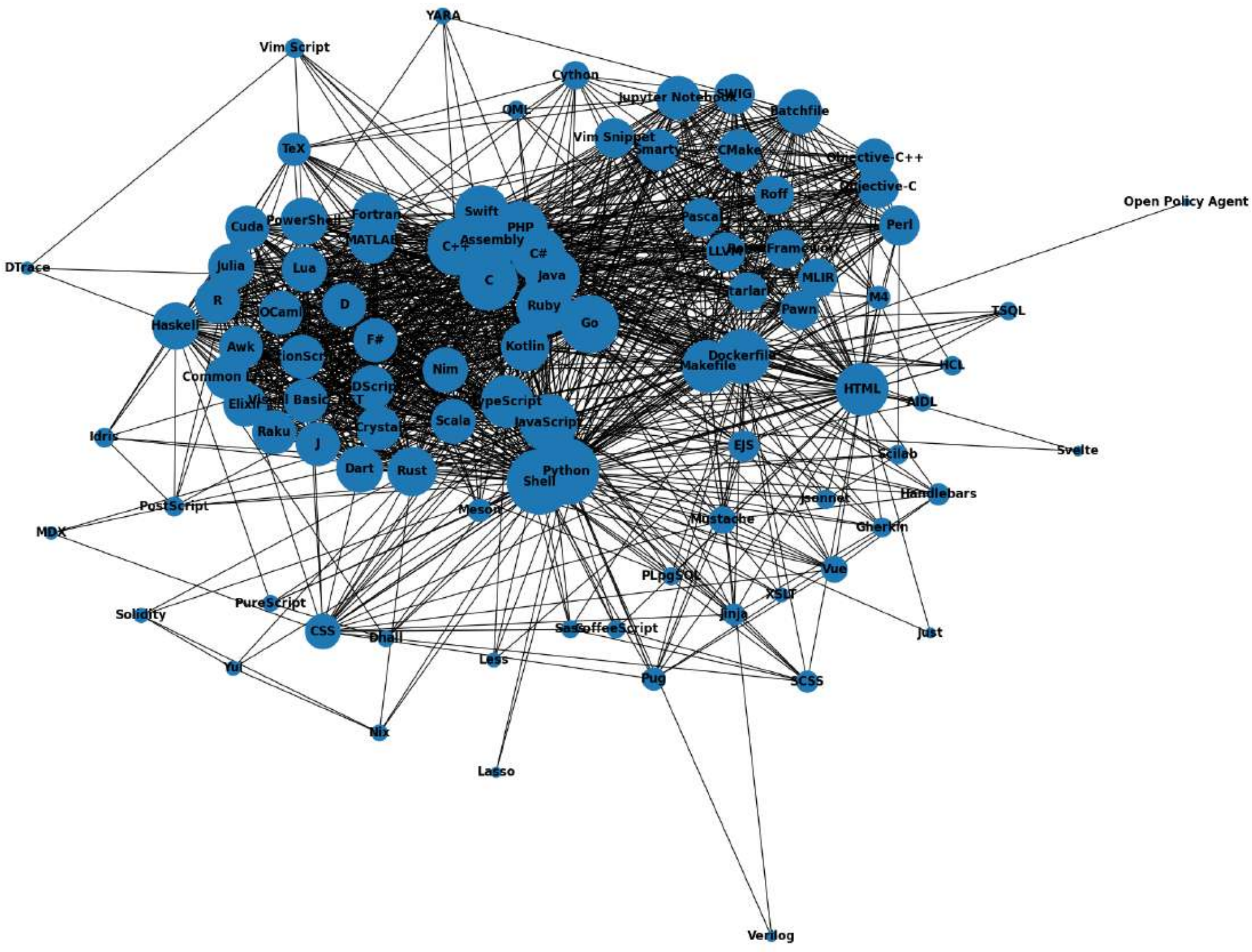
[9] Kruskal, J. B, “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem” In Proceedings of the American Mathematical Society (Vol. 7, No. 1, pp. 48-50), 1956

[10] A. S. Tanenbaum e H. Bos, ”Modern Operating Systems” (Vol. 4), 2014

[11] R. A. C. Moreira, ”A.E.I.L.P.”, GitHub, 2023. [Online]. Disponível: https://github.com/Guiliard/A.E.I.L.P..git


HTML
TeX
JavaScript
Shell
Emacs Lisp
Python
CSS
Cython
Makefile
Stata
Perl
Roff
C


Cuda
Objective-C++
Go
Objective-C
MATLAB
M4
CMake
Ruby
Java
Jupyter Notebook
PowerShell
C
C++
Perl
C#
SaltStack
Perl 6
Nix
Assembly
PHP
CSS
Lua
Protocol Buffer
Verilog
Module Management
SAS
Scala
Visual Basic
Dockerfile
Starlark
HTML
SMT
Roff
Nginx
SystemVerilog
Makefile
Lex
JavaScript
Jsx
VHDL
Shell
Yacc
Less
Coq
Python
SCSS
Haskell
CoffeeScript
TypeScript
Vim Script
PostScript
R
Cython
HCL
GDB
GLSL
Just
Rust
Pascal
Jinja


YARA
Vim Script
Cython
QML
Jupyter Notebook
SWIG
Batchfile
Vim Snippet
Smarty
CMake
Objective-C++
Objective-C
TeX
Roff
Open Policy Agent
Fortran
PowerShell
Swift
PHP
Assembly
Cuda
MATLAB
C++
Pascal
Perl
C#
LLVM
DTrace
Julia
Lua
Java
MLIR
C
Starlark
R
D
Ruby
M4
OCaml
Pawn
TSQL
Haskell
Go
F#
Awk
Kotlin
Makefile
Dockerfile
HCL
Nim
HTML
AIDL
Elixir
Scala
JavaScript
Raku
Crystal
Idris
EJS
Scilab
Svelte
J
Python
Dart
Rust
Shell
Jsonnet
Handlebars
PostScript
Meson
Mustache
MDX
Gherkin
Vue
PLpgSQL
PureScript
XSLT
Solidity
Jinja
CSS
Dhall
CoffeeScript
Just
Yul
Less
Pug
SCSS
Nix
Lasso
Verilog