*Subject Section*

# PETAL: A Python tool for deep analysis of biological pathways

Giuseppe Sgroi[1], Giulia Russo[2,*] and Francesco Pappalardo[2]

[1]Department of Mathematics and Computer Science, University of Catania, V.le A. Doria, 6, 95125 Catania, Italy, [2]Depatment of Drug Sciences, University of Catania, V.le A. Doria 6, 95125 Catania, Italy.

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

## Abstract

**Summary:** Although several bioinformatics tools have been developed to examine signaling pathways, little attention has been given to ever long-distance crosstalk mechanisms. Here, we developed PETAL, a Python tool that automatically explores and detects the most relevant nodes within a KEGG pathway, scanning and performing an in-depth search. PETAL can contribute to discovering novel therapeutic targets or biomarkers that are potentially hidden and not considered in the network under study.

**Availability:** PETAL is a freely available open-source software. It runs on all platforms that support Python3. The user manual and source code are accessible from https://github.com/Pex2892/PETAL.

**Contact:** giuseppe.sgroi@unict.it

## 1 Introduction

Several web-based applications for pathway analysis and biomarker identification (Cirillo *et al.*, 2017) have been developed to extract more complete and interpretable features from the molecular and biochemical mechanisms among individual genes, proteins or metabolites in predefined pathways present in biological databases (e.g. Kyoto Encyclopedia of Genes and Genomes (KEGG) (Kanehisa, 2004). Here, we present ParallEl paThways AnaLyzer (PETAL), a tool that provides a better usage of KEGG through the employment of new specifically developed open-source tools and Python libraries (Lindstrom, 2005), such as Pandas[1] and Joblib[2]. In particular, PETAL allows users to find hidden interactions among significant proteins belonging to the same pathway and other proteins within possible linked pathways. Web-tools like STRING (Szklarczyk *et al.*, 2015), Cytoscape (Franz *et al.*, 2015) and Reactome, with its browser pathway function (Fabregat *et al.*, 2018) have added valuable features in terms of visualization of complex networks, and analysis of pathway knowledge to support basic research. However, PETAL adds the possibility to search in-depth for ancestor and descendent nodes of a specific target gene, making this task faster in terms of performance. PETAL is potentially able to find pathways that are distant from the ones containing the targets of interest. Moreover, PETAL includes a scalable and parallelized engine with the ability to easily add new functionalities and specific modules to simplify and automate the discovery of new therapeutic targets or biomarkers. Inspired by Palumbo's work and related tool MapReduce (Palumbo *et al.*, 2019), we propose a more efficient alternative solution in terms of computational time and performance. Compared to MapReduce, PETAL strongly optimizes the in-depth search. It employs a type of data structure called data frame containing specific information about the initial gene and the ending gene used to perform the step-by-step in-depth analysis. This RAM-saved data frame performs much better in terms of reading/writing access and allows an easier implementation when used in HPC environment. Conversely, MapReduce accesses the main memory more frequently at every single step, slowing down the search. Another significant difference is the information extracted from each analysed pathway and gene: MapReduce only tracks the connections found during the analysis, while PETAL holds the pathways of origin, the protein interaction and its biological function, and the

[1] https://pandas.pydata.org/

[2] https://joblib.readthedocs.io/en/latest/

connections found. Moreover, during the analysis the number of occurrences is calculated for each connection. Finally, PETAL provides a graphical user interface allowing the user to navigate through an interactive tree containing all the information found during the search. The main advantages of the tool can be summarized in five points: *i)* a better performing breadth-first search (BFS); *ii)* the decreased main memory request during the analysis, avoiding text files management; *iii)* the calculation of occurrences during the analysis (i.e., the number of times that a determined pathway has been found in all the pathways retrieved from KEGG during the search process between the starting gene and the ending one); *iv)* a user-friendly interface with more information shown and *v)* the possibility to save pathways locally. To get a parallel working version, Joblib has been employed. Joblib is a set of tools that lets Python manage parallel computing in an optimized way to be fast and robust on extensive data. PETAL output is made available through the d3.js library (Bostock *et al.*, 2011). D3.js is a JavaScript library developed to display data dynamically and interactively, starting from some organized numeric data that combine HTML5, Scalable Vector Graphics (SVG), and Cascading Style Sheets (CSS). It is worth mentioning that PETAL GUI was not developed entirely from scratch but from the "Radial Tree" project developed by Wm Leler (https://gist.github.com/wmleler/a734fb2bb3319a2cb386). It represents a viewer of interactive radial trees in which users can navigate and explore all the available information obtained during the search. To this aim, the GUI includes specific commands that allow users to manage the output tree.

## 2 Design and function

PETAL uses the BFS search logic to carry out the breadth analysis, which starts with the initial retrieving of data, in particular the biological pathways present in KEGG. The analysis requires the following input parameters: pathway name, starting gene, and maximum search depth. In this way, it is possible to discover fewer common pathways and show the number of times they appear in other biological pathways. Here, the advantage consists of automatically and quickly obtaining the results, saving time and decreasing complexity. This process would have requested much more time if conducted manually. Moreover, a graphical interface allows the exploration of the output tree, showing all the connections among genes in different depths, the number of occurrences, and other pathway information.

### 2.1 Reading the configuration file

The configuration file is read in INI format[3] through the "read_config()" method, a structure similar to what is found in Microsoft Windows INI files, containing a set of mandatory and optional parameters for starting the analysis:

- #CPU (optional): <integer>, sets the number of CPUs available for the analysis. If this parameter is set to zero or its value overcomes the number of installed CPUs, it is automatically set to the maximum number.
- Pathway (mandatory): <string>, represents the biological pathway from which the analysis starts.
- Gene (mandatory): <string>, represents the gene contained in the pathway previously selected from which the analysis begins. If the inserted gene is not contained in the biological pathway, an error is issued.
- Depth (mandatory): <integer>, sets the maximum depth of the analysis.

[3] https://docs.python.org/3/library/configparser.html

### 2.2 Checking for pathway updates

The web page https://www.genome.jp/kegg/docs/upd_map.html, available on KEGG, lists the names of the updated pathways and the type of adjustment. The method "download_update_pathway_html()" checks if any pathways have been updated, and eventually it downloads the new XML file; if there are no updates, the process stops and proceeds to the next step.

### 2.3 Preparation of data structures

In this step, the global DataFrame structure is created. It contains all the connections found during the analysis, grouped by depth. Each row shows the following data collection and information:

1. "deep": shows the depth of the connection between the two genes (e.g., 1);
2. "name_father": indicates the name of the initial gene (e.g., MAPK1);
3. "hsa_father": specifies the name of the initial gene in hsa format (e.g., hsa:5594 hsa:5595);
4. "name_son": indicates the name of the ending gene (e.g., EGFR);
5. "hsa_son": specifies the name of the ending gene in hsa format (e.g., has:1956 has:2064);
6. "url_kegg_son": shows the address where it is possible to retrieve the list of paths in which the gene of interest is present (e.g., https://www.kegg.jp/dbget-bin/www_bget?hsa:1956+hsa:2064);
7. "relation": shows the protein interaction between the two genes (e.g., PPre§§PPrel§§PPrel);
8. "type_rel": represents the nature of the biochemical interaction such as a phosphorylation reaction (e.g., activation//indirect effect§§activation//indirect effect§§activation//phosphorylation//indirect effect);
9. "pathway_of_origin": contains the list of pathways whose ending gene is present (e.g., has04921§§has04928§§has05205);
10. "fullpath": specifies the complete hierarchy from the initial gene to the ending gene found (e.g., MAPK1/EGFR);
11. "occurrences": shows the number of occurrences (e.g., 6).

### 2.4 Analysis

The analysis is performed in parallel through the Joblib library. To speed-up the execution when reading the XML files compressed files by gzip are used. The analysis deals with a depth equal or superior to 1. In the former case, the analysis takes care of the initial gene, while in the latter one, a list of genes is considered. Precisely, the analysis consists of the following sub-phases:

- *Depth 1*

a. Download of the XML file of the selected pathway.
b. Reading of the XML file and extraction of those genes directly connected to the one selected in the configuration file.
c. Addition of the genes found in the data frame.
d. Creation of a list of pathways in which the selected gene is present.

e. Processing each pathway in the list in parallel, obtaining from each process a list of found genes directly connected to the selected gene.
f. Each process, upon its completion, provides the results in the data frame.

- *Depth 2, ..., n*

a. Extraction of the descendent genes previously found and saving them in a list.
b. Processing of each gene in the list through steps d), e), and f) of *Depth 1* sub-phase.

### 2.5 Generation and display of the output

A JSON file must be created to view the tree by inserting all the links saved in the global data frame after the analysis. Figure 1 shows a screenshot of PETAL GUI: on the left, one can see the connection tree and the command toolbox. On the right, one can see the information of a specific gene divided by 1) whole path, 2) depth, 3) number of occurrences, and 4) related organism code (e.g., "hsa"), which identifies the gene accordingly to KEGG nomenclature.

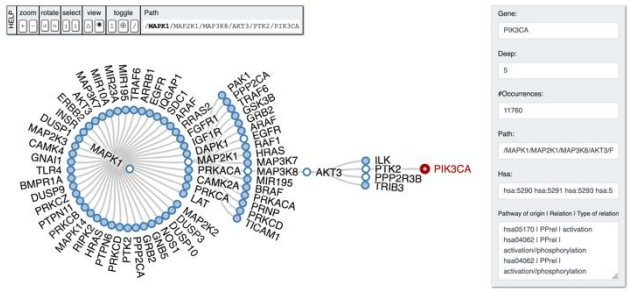


**Fig. 1. An output tree example (MAPK1 pathway) obtained by running the algorithm with a depth equal to 5**. The figure depicts an interactive example of the output tree generated by analysing the biological pathway "MAPK". The starting gene is represented by MAPK1 (alias ERK) and the depth search level is equal to 5. As one can see, all the genes in output are classified by their depth; the gene in red (PIK3CA) represents the ending gene level reached by the analysis. The box on the right shows specific information related to the pathway under study for every single gene. The upper panel represents the graphical user interface control buttons.

## 3 Performance and simulation results

PETAL tests were performed on a server equipped with 8 Intel Xeon E5 CPUs at 2.40 GHz and 64 Gigabytes of RAM with different combinations of depth levels ranging from 1 to 5, using 1, 2, 4 and 8 CPUs. Each step is performed starting from the initial gene. As a working example, we considered MAPK1 genes (hsa5594 + hsa5595) contained in MAPK pathway (hsa04010) (Morrison, 2012), as reported in KEGG. MAPK1 represents one of the most important biomarkers of response to protein kinase inhibitors associated with cellular proliferation and differentiation in several types of cancer, such as melanoma and thyroid cancer (Gianì *et al.*, 2019; Pappalardo *et al.*, 2016). As shown in the resulting tree figure 1, running the algorithm with a depth equal to 5 and starting from MAPK1 gene, a generation of four output genes (MAP2K1, MAP3K8, AKT3, PTK2 and PIK3CA) is obtained, moving forward to their depth gene level. At level 3, the output graph pointed out the presence of MAP3K8 gene as one of the possible descendent genes involved in the modulation of MAPK1. This latter seems not directly linked to MAP3K8 as MAPK1 pathway (hsa04010) as reported in KEGG. Hence, from a biological point of view, this depth search helps to retrieve certain interesting genes potentially hidden and involved in important biological and cellular process.

Results in Table 1 show the number of connections found, the level of depth search, and the total number of times (expressed in seconds) necessary to complete the analysis without and with locally available data. It is worth mentioning that there is no improvement during the analysis at depth 1, because there is a constant performance time. Interesting improvements have been observed starting from depth 2 using 8 CPUs. This is due to the fact that the number of connections to be processed becomes ever greater in terms of time and computational cost. As one can see, when data are not locally available, the total time of each run is much higher. This fact is dependent on three reasons: i) the quality and the speed of internet connection; ii) the number of CPUs assigned for the running of the process, and iii) the time required by the server to download the XML file of the pathway.

**Table 1.** Computational efforts expressed in second required to complete the analysis at different level of depth

| Depth | #Connections | 1 CPU | 2 CPUs | 4 CPUs | 8 CPUs |
|---|---|---|---|---|---|
| 1 | 45 | 108/3 | 57/3 | 31/ 3 | 20/2 |
| 2 | 406 | 295/38 | 209/26 | 162/20 | 145/18 |
| 3 | 1669 | 977/248 | 818/170 | 762/137 | 716/122 |
| 4 | 4140 | 2544/1040 | 2109/707 | 1963/587 | 1890/544 |
| 5 | 7731 | 5374/3121 | 4385/2302 | 4031/2052 | 3899 / 1892 |

## 4 Conclusions

PETAL is an open-source tool that offers a wide range of features to easily explore and detect the most relevant nodes within a KEGG pathway in a significantly shorter time when compared to manual analysis. Potential users can find supplementary information and source code freely available on GitHub repository. Future PETAL releases will allow the user to access other pathway databases than KEGG (i.e., Reactome and WikiPathways).

*Conflict of Interest:* none declared.

## References

Bostock,M. *et al.* (2011) $D^3$ Data-Driven Documents. *IEEE Trans. Vis. Comput. Graph.*, **17**, 2301–2309.

Cirillo,E. *et al.* (2017) A review of pathway-based analysis tools that visualize genetic variants. *Front. Genet.*

Fabregat,A. *et al.* (2018) The Reactome Pathway Knowledgebase. *Nucleic Acids Res.*

Franz,M. *et al.* (2015) Cytoscape.js: A graph theory library for visualisation and analysis. *Bioinformatics*.

Gianì,F. *et al.* (2019) Computational modeling reveals MAP3K8 as mediator of resistance to vemurafenib in thyroid cancer stem cells. *Bioinformatics*, **35**, 2267–2275.

Kanehisa,M. (2004) The KEGG resource for deciphering the genome. *Nucleic Acids Res.*, **32**, 277D – 280.

Lindstrom,G. (2005) Programming with Python. *IT Prof.*, **7**, 10–16.

Morrison,D.K. (2012) MAP kinase pathways. *Cold Spring Harb. Perspect. Biol.*, **4**.

Palumbo,G.A.P. *et al.* (2019) A MapReduce tool for in-depth analysis of KEGG pathways: identification and visualization of therapeutic target candidates. In, *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, pp. 2157–2162.

Pappalardo,F. *et al.* (2016) Computational Modeling of PI3K/AKT and MAPK Signaling Pathways in Melanoma Cancer. *PLoS One*, **11**, e0152104.

Szklarczyk,D. *et al.* (2015) STRING v10: Protein-protein interaction networks, integrated over the tree of life. *Nucleic Acids Res.*