

# Applying the Pathway Activity Profiling - PAPI

Raphael Aggio

October 30, 2018

## Introduction

This document describes how to use the Pathway Activity Profiling - *PAPi*. *PAPi* is an R package for predicting the activity of metabolic pathways based solely on metabolomics data. It relates metabolites' abundances with the activity of metabolic pathways. See Aggio, R.B.M; Ruggiero, K. and Villas-Boas, S.G. (2010) - Pathway Activity Profiling (*PAPi*): from metabolite profile to metabolic pathway activity. Bioinformatics.

## 1 Step 1 - Building a local KEGG database: buildDatabase

*PAPi* makes use of the information available at the Kyoto Encyclopedia of Genes and Genomes - KEGG database (<http://www.genome.jp/kegg/>). *PAPi* can be applied in two ways: online and offline. When applying *PAPi* online, an internet connection is required to collect online biochemical data from KEGG database. On the other hand, when applied offline, *PAPi* makes use of a local database and no internet connection is required. As KEGG database is constantly updated, *PAPi* may produce different results when applied online on different days. Offline, *PAPi* produces always the same results when using the same local database. *PAPi* is significantly faster when applied offline. As default, *PAPi* brings a local database and the function `buildDatabase` can be used to build new local databases. `buildDatabase` uses the internet connection to create and install new databases inside of the folder `R.home("library/PAPi/data/databases/")`. If `save = TRUE`, the new database is also installed in a folder defined by the user. The local database consists on two CSV files containing the required data about metabolites and metabolic pathways. `buildDatabase` may take up to ten hours depending on the speed of the internet connection. We highly recommend to build a local database once and do all the required analysis using the same database. Doing so, the results obtained from *PAPi* can be reproduced at anytime and the time required for each analysis is enormously faster.

## 2 Step 2 - Adding KEGG codes to metabolomics data: addKeggCodes

We consider a typical metabolomics data set: a data frame containing the name of identified metabolites in the first column and their abundances in the different samples in the following columns. Bellow you have an example of a metabolomics data set.

```
> library(PAPi)
> library(svDialogs)
> data(metabolomicsData)
> print(metabolomicsData)
```

	Names	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6
1	Replicates	cond1	cond1	cond1	cond2	cond2	cond2
2	Glucose	0.8	0.6	0.7	1.2	1.1	1.5
3	Fructose 6-phosphate	0.3	0.2	0.4	0.6	0.7	0.7
4	Glyceraldehyde 3-phosphate	1.1	1.5	1.2	<NA>	<NA>	<NA>
5	Glycerone phosphate	1.2	1.5	1.3	0.2	0.3	0.2
6	3-Phospho-D-glycerate	0.2	0.3	0.5	1.1	1	0.9

For applying *PAPi*, the name of compounds in the metabolomics data set must be substituted by their respective KEGG codes. This process can be performed manually, however, it may be considerable time consuming according to the size of the input data. The function `addKeggCodes` automates this process using a KEGG codes list, which is a data frame containing all potentially identifiable compound and their respective KEGG codes. See bellow:

```
> data(keggLibrary)
> print(keggLibrary)
```

	kegg	Name
1	C00031	Glucose
2	C05345	Fructose 6-phosphate
3	C00118	Glyceraldehyde 3-phosphate
4	C00111	Glycerone phosphate
5	C00197	3-Phospho-D-glycerate
6	absent	Citrate

The KEGG codes list can be built as a CSV file containing a list of KEGG codes in the first column and their respective compound names in the second column, following the same format as showed above. Alternatively, the data frame `data(keggLibrary)` can be used. Ideally, the KEGG codes list should contain every compound potentially identifiable by the protocol you use to analyze and identify metabolites. For example, I have in my KEGG codes list the name of every compound present in my GC-MS library. The KEGG code of a compound can be searched at KEGG website: <http://www.genome.jp/kegg/>, however, if the argument `addCodes = TRUE`, the function `addKeggCodes` will automatically recognize if you have missing compounds - compounds in your input data that are not present in your KEGG codes list. Then, it will

ask if you would like to add those compounds to the KEGG codes list in use. If you click on "yes", `addKeggCodes` will automatically search the KEGG database (using the INTERNET connection) for missing compounds. For each missing compound, `addKeggCodes` will present to you a list with all the potential matches from KEGG database. If the right compound is in the list, you just have to click on the compound and click on "Ok". The KEGG code and the name of this respective compound will be automatically added to the bottom of your KEGG codes list. On the other hand, if the desired compound is not in the list presented to you, you have two options: go to the end of the list and click on "SKIP"; or go to the end of the list and click on "OTHER", which will open a new dialog box allowing you to manually add the respective KEGG code. If the compounds' names in the input data are similar to the names used by KEGG database, `addKeggCodes` will find the KEGG code of most compounds. After looking for KEGG codes, `addKeggCodes` gives you the option to save the new KEGG codes list as CSV file and generates a data frame as the input data, however, now with KEGG codes instead of compounds' names. Here is an example of how to use `addKeggCodes` and the result it generates:

```
> print(metabolomicsData)
```

	Names	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6
1	Replicates	cond1	cond1	cond1	cond2	cond2	cond2
2	Glucose	0.8	0.6	0.7	1.2	1.1	1.5
3	Fructose 6-phosphate	0.3	0.2	0.4	0.6	0.7	0.7
4	Glyceraldehyde 3-phosphate	1.1	1.5	1.2	<NA>	<NA>	<NA>
5	Glycerone phosphate	1.2	1.5	1.3	0.2	0.3	0.2
6	3-Phospho-D-glycerate	0.2	0.3	0.5	1.1	1	0.9

```
> print(keggLibrary)
```

	kegg	Name
1	C00031	Glucose
2	C05345	Fructose 6-phosphate
3	C00118	Glyceraldehyde 3-phosphate
4	C00111	Glycerone phosphate
5	C00197	3-Phospho-D-glycerate
6	absent	Citrate

```
> AddedKegg <- addKeggCodes(
+   metabolomicsData,
+   keggLibrary,
+   save = FALSE,
+   addCodes = TRUE
+ )
```

```
[1] "Data frame loaded..."
```

```
[1] "keggCodes - Data frame loaded..."
```

```
[1] "Every compound in the inputData was found in the keggCodes library."
```

```
[1] "The final data frame was not saved because the argument save was set as FALSE"
```

```
> print(AddedKegg)
```

	Name	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6
1	Replicates	cond1	cond1	cond1	cond2	cond2	cond2
2	C00197	0.2	0.3	0.5	1.1	1	0.9
3	C05345	0.3	0.2	0.4	0.6	0.7	0.7
4	C00031	0.8	0.6	0.7	1.2	1.1	1.5
5	C00118	1.1	1.5	1.2	<NA>	<NA>	<NA>
6	C00111	1.2	1.5	1.3	0.2	0.3	0.2

### 3 Step 3 - Applying PAPI: papi

Once the names of metabolites were substituted by their respective KEGG codes, the input data is ready to be analyzed by the function `papi`. Here is an example of how to use `papi` and the results it generates:

```
> data(papiData)
> print(papiData)
```

	Name	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6
1	Replicates	cond1	cond1	cond1	cond2	cond2	cond2
2	C00197	0.2	0.3	0.5	1.1	1	0.9
3	C05345	0.3	0.2	0.4	0.6	0.7	0.7
4	C00031	0.8	0.6	0.7	1.2	1.1	1.5
5	C00118	1.1	1.5	1.2	<NA>	<NA>	<NA>
6	C00111	1.2	1.5	1.3	0.2	0.3	0.2

```
> #papiResults <- papi(papiData, save = FALSE, offline = TRUE, localDatabase = "default")
> data(papiResults)
> head(papiResults)
```

	pathwayname	Sample1	Sample2	Sample3	Sample4
1	Replicates	cond1	cond1	cond1	cond2
22	Bile secretion	140	105	122.5	210
10	Methane metabolism	92	120	100	16
20	ABC transporters	79.2	59.4	69.3	118.8
13	Pentose and glucuronate interconversions	63.25	82.5	68.75	11
12	Nicotinate and nicotinamide metabolism	56.4	70.5	61.1	9.4
	Sample5	Sample6			
1	cond2	cond2			
22	192.5	262.5			
10	24	16			
20	108.9	148.5			
13	16.5	11			
12	14.1	9.4			

When `offline = TRUE`, PAPI is performed using a local database. When `localDatabase = "choose"`, a list of all installed local databases is presented to the user to choose which one to be used.

## 4 Step 4 - Applying t-test or ANOVA: papiHtest

`papiHtest` performs a t-test or ANOVA on the input data, which is a data frame produced by the function `papi`. For identifying to which experimental condition each sample belongs, the first row of the input data may receive the value "Replicates" in the first cell of the first column and a character string indicating to which experimental condition each sample belongs in the following columns, such as described below:

```
> data(papiResults)
> head(papiResults)
```

	pathwayname	Sample1	Sample2	Sample3	Sample4
1	Replicates	cond1	cond1	cond1	cond2
22	Bile secretion	140	105	122.5	210
10	Methane metabolism	92	120	100	16
20	ABC transporters	79.2	59.4	69.3	118.8
13	Pentose and glucuronate interconversions	63.25	82.5	68.75	11
12	Nicotinate and nicotinamide metabolism	56.4	70.5	61.1	9.4
	Sample5	Sample6			
1	cond2	cond2			
22	192.5	262.5			
10	24	16			
20	108.9	148.5			
13	16.5	11			
12	14.1	9.4			

Alternatively, `papiHtest` will ask the user to indicate the samples belonging to each experimental condition. It is all interactive. The argument `StatTest` of `papiHtest` is used to determine if a t-test or an ANOVA should be performed. If `StatTest` = "T-TEST", "T-test", "t-test", "t-TEST", "t" or "T", a t-test is performed. If `StatTest` = "ANOVA", "Anova", "anova", "A" or "a", a ANOVA is performed. If `StatTest` is missing, t-test will be performed for input data sets showing two experimental conditions and ANOVA will be performed for input data sets showing more than two experimental conditions. `papiHtest` can be applied as follows:

```
> head(papiResults)
```

	pathwayname	Sample1	Sample2	Sample3	Sample4
1	Replicates	cond1	cond1	cond1	cond2
22	Bile secretion	140	105	122.5	210
10	Methane metabolism	92	120	100	16
20	ABC transporters	79.2	59.4	69.3	118.8
13	Pentose and glucuronate interconversions	63.25	82.5	68.75	11
12	Nicotinate and nicotinamide metabolism	56.4	70.5	61.1	9.4
	Sample5	Sample6			
1	cond2	cond2			
22	192.5	262.5			
10	24	16			
20	108.9	148.5			
13	16.5	11			
12	14.1	9.4			

```

> ApplyingHtest <- papiHtest(
+   papiResults,
+   save = FALSE,
+   StatTest = "T"
+ )
> head(ApplyingHtest)

```

	pathwayname	Sample1	Sample2	Sample3	Sample4
1	Replicates	cond1	cond1	cond1	cond2
12	Carbon fixation in photosynthetic organisms	26.45	34.5	28.75	4.6
7	Clavulanic acid biosynthesis	11	15	12	<NA>
23	Fructose and mannose metabolism	41.6	51.2	46.4	19.2
33	Inositol phosphate metabolism	48.3	63	52.5	8.4
13	Methane metabolism	92	120	100	16

  

	Sample5	Sample6	pvalues
1	cond2	cond2	bonferroni
12	6.9	4.6	0.0391647314284331
7	<NA>	<NA>	0
23	24	21.6	0.0335649923259184
33	12.6	8.4	0.0391647314284331
13	24	16	0.0391647314284331

## 5 Step 5 - Generating PAPI graph: papiLine

The results produced by PAPI can be automatically plotted in a line graph using the function `papiLine`. The input data is a data frame such as produced by `papi` and `papiHtest`. `papiLine` was developed to automatically detect graphical parameters, however, some particular data sets may end up in graphs with the legend in the wrong position or axis labels too small. For this reason, `papiLine` has 20 arguments that allow the user to customize most of the graphical parameters. If `save = TRUE`, a png file will be saved in a folder defined by the user. If `save = FALSE`, a new window will pop up with the graph and the user can use the R options for saving the graph to a PDF file. Here is an example of how to use `papiLine`:

```

> head(papiResults)

```

	pathwayname	Sample1	Sample2	Sample3	Sample4
1	Replicates	cond1	cond1	cond1	cond2
22	Bile secretion	140	105	122.5	210
10	Methane metabolism	92	120	100	16
20	ABC transporters	79.2	59.4	69.3	118.8
13	Pentose and glucuronate interconversions	63.25	82.5	68.75	11
12	Nicotinate and nicotinamide metabolism	56.4	70.5	61.1	9.4

  

	Sample5	Sample6
1	cond2	cond2
22	192.5	262.5
10	24	16
20	108.9	148.5

```
13      16.5      11
12      14.1      9.4
```

```
> papiLine(
+     papiResults,
+     relative = TRUE,
+     setRef.cond = TRUE,
+     Ref.cond = "cond1",
+     save = FALSE
+ )
```

```
[1] "Data frame loaded..."
```