PathWave: short manual

Version 1.0

February 4th, 2009

This manual gives a short introduction into the usage of the R package *PathWave*. *PathWave* enables the user to easily analyse gene expression data considering the topology of metabolic pathways as provided by KEGG.

The package can be downloaded from http://www.ichip.de/software/pathwave.html

This web page also provides an executable R script, 'usePathWave.R'. This script is an example how to analyse the neuroblastoma data of our case study (Schramm et al. submitted). For that purpose, it combines the methods of the package *PathWave*. The script is explained in more detail at the end of this manual.

To run the example analysis via script 'usePathWave.R' a data set has to be downloaded from the same web page. The data is bundled into the zipped file PathWaveFilesFeb04_2009.
Downloading and unzipping the data files for our neuroblastoma case study, PathWaveFilesFeb04_2009 creates a new folder PathWaveFilesFeb04_2009 containing 3 folders:

1. Folder “Data”:

agilentIDToEntrezGene.tab:	the mapping from Agilent Ids to Entrez Gene Ids
Neuroblastoma_vsn_expr_stage1_4amp.tab:	the normalized expression data of 65 stage 1 and 19 stage 4 (with amplification of MYCN) tumors
Neuroblastoma_vsn_expr_stage1_4amp.class:	the class information for the samples 1: stage 1, 4amp: stage 4 amplified MYCN
2. Folder “OptLogFiles”:	contains results of the optimization process for every KEGG pathway (not necessarily needed for further analysis)
3. Folder “XML”:	KEGG XML files from February 4th, 2009

- The gene expression data is in the format of a tab delimited table with columns being the samples, rows the gene probes
- The class information is stored as a white space delimited matrix with three columns. These are the numbering of the patients, the sample Ids (as used in Oberthuer et al. and stored in Array Express at http://www.ebi.ac.uk/arrayexpress , experiment accession number E-TABM-38) and the class names of each sample.
- Sample Ids in the gene expression data correspond to the sample Ids in the class information

In the following the installation of *PathWave* for Windows and Linux platforms is described.
How these methods can be used to analyse a data set is then described in the following by explaining how to run the script 'usePathWave.R' with our case study (Schramm et al. submitted) as an example data set, followed by a description of the script 'usePathWave.R'. Finally, an overview over the methods bundled into the package *PathWave*.is given at the end of this manual.

Installing *PathWave* under Windows:

R can be downloaded from http://www.r-project.org/ and installed as mentioned there.

For using *PathWave* R version 2.6.0 or higher is required

Before using *PathWave* some additional R packages have to be installed. After starting R this can easily be done. To install packages from Bioconductor (http:// www.bioconductor.org) type into the R prompt:

```
source("http://bioconductor.org/biocLite.R")
biocLite("Biobase")
biocLite("RCurl")
biocLite("genefilter")
```

Additional packages are installed by clicking on "Packages", "Install package(s)", selecting a mirror and then selecting the packages that should be installed. *PathWave* requires:

e1071
multtest
waveslim
evd
XML

Now everything is prepared to install *PathWave.*

Download PathWave_1.0.zip. Do not unzip it.

PathWave is installed by clicking on "Packages", "Install package(s) from local zip files..." and selecting the file PathWave_1.0.zip.

For help see also http://cran.r-project.org/bin/windows/base/rw-FAQ.html

After installation, the package *PathWave* can be loaded into R by

```
library(PathWave)
```

Help pages for the methods of *PathWave* are accessible by typing "?" and the name of the method:

```
?pathWave
?pwKEGGxml
?pwAdjMatrices
?pwLPFiles
?pwOptGrids
```

Installing *PathWave* under Linux:

Before using *PathWave* some additional R packages have to be installed. After starting R this can easily be done. To install packages from Bioconductor (http:// www.bioconductor.org) type into the R prompt:

```
source("http://bioconductor.org/biocLite.R")
biocLite(“Biobase”)
biocLite(“RCurl”)
biocLite(“genefilter")
```

Additional packages are installed by typing:

```
install.packages(“nameOfPackage”)
```

The packages that should be installed, are:

e1071
multtest
waveslim
evd
XML

Now everything is prepared to install *PathWave*.

Download PathWave_1.0.tar.gz. Do not unzip it.

The R package *PathWave* will be installed locally. Therefore, create a folder where *PathWave* should be installed, e.g. ~/myRlib/

Copy PathWave_1.0.tar.gz into that folder and install it

```
install.packages(“~/myRlib/PathWave_1.0.tar.gz”, repos=NULL, lib="~/myRlib/")
```

Loading package *PathWave*:

For loading the package *PathWave* R needs to know where to search for the package. Create a variable and store the path in which *PathWave* was installed

```
myRlib=”~/myRlib/”
```

Now load *PathWave*

```
library(PathWave, lib.loc=myRlib)
```

The help pages for the methods of *PathWave* are accessible by typing "?" and the name of the method:

```
?pathWave
?pwAdjMatrices
?pwKEGGxml
?pwLPFiles
?pwOptGrids
```

run *PathWave* on our case study

Before conducting the example analysis with *PathWave* some preparations have to be made. First, the executable R script 'usePathWave.R' has to be downloaded.

The file 'usePathWave.R' can be opened by any available editor. It is a script for an example how to analyse the neuroblastoma data. For that purpose, 'usePathWave.R' combines the methods of the package *PathWave*. The script is explained in more detail at the end of the manual.

To use 'usePathWave.R', it has to be in the working directory of R. To see what your actual working directory is, type

```
getwd()
```

If the actual working directory is not the directory where 'usePathWave.R' was downloaded to, you have to change it. You can either copy the script file into the actual working directory or you change the working directory itself. This can be done by using setwd() as described in the following.

To set the working directory to the directory in which 'usePathWave.R' is, use setwd() e.g.
setwd("../Desktop/")

```
setwd("../myWorkingDirectory/")
```

Secondly the zipped example data set PathWaveFilesFeb04_2009 has to be downloaded and unzipped.

After this is done, start R and load *PathWave* as described above.

As we would like to use the downloaded data, R needs to know where this data is stored. Therefore, one variable has to be set to run 'usePathWave.R' successfully.

Unzipping PathFilesFeb04_2009 results in a folder PathFilesFeb04_2009. The directory path to this folder should accordingly be stored in a variable:

Define variable **myPathWavePath** by

```
myPathWavePath = "../PathWaveFilesFeb04_2009/"
```

for which the dots denote the directory path to folder PathWaveFilesFeb04_2009/ and are to be filled with the correct directory values

CAUTION UNDER WINDOWS:

R needs slashes “/” as input and does not accept the input of backslashes “\”. Thus, you have to exchange all “\” for "/" !
The directory path should look like C:/../PathWaveFilesFeb04_2009/

Now we are prepared to start the analysis. Type

source(“usePathWave.R”)

The analysis may take a while.

After successfully running 'usePathWave.R' a table with crucial information should be returned corresponding to table 1 of the publication (Schramm et al., submitted)

The columns denote the
1. KEGG Id,
2. Number of differentially regulated reactions,
3. Number of all reactions,
4. Number of up-regulated reactions,
5. Number of down-regulated reactions,
6. p-value of the pathway, calculated by *PathWave* (using wavelet transforms)

Furthermore, the variable **result** can be accessed with the stored results. E.g.

names(result)

shows the KEGG Ids of the enriched pathways, e.g.

result$hsa00251$reaction.regulation

shows the regulation pattern of glutamate metabolism (KEGG Id hsa00251)

In the documentation of method *pathWave()* above, the structure of variable **result** is explained in more depth.

What does 'usePathWave.R' do?

The R script 'usePathWave.R' combines the methods of *PathWave* demonstrating how to use the package on a dataset.

In a first step the gene expression data has to be mapped onto the reactions of every KEGG pathway. Therefore, it is necessary to know which gene (or combination of genes) encodes for which enzyme. KEGG supplies us with this information. Extracting this information can be done by using the method *pwKEGGxml()*. The method takes as argument the path (for downloaded files) or the Internet address where KEGG's XML files are stored (e.g. ftp://ftp.genome.jp/pub/kegg/xml/organisms/hsa/).

For this analysis we have to set the path in which the downloaded XML files (version form February 4th 2009) are kept and parse these files (this may take a while).
Remember: We have set the variable **myPathWavePath** in the beginning (before using 'usePathWave.R').

```
keggXmlPath=paste(myPathWavePath,"XML/",sep="/")
keggXml=pwKEGGxml(keggXmlPath)
```

As a next step the assignment of Entrez Gene Ids to the probe Ids of the chip is read in as a matrix. This information is stored in the table agilentIDToEntrezGene.tab in folder Data/ of the downloaded PathWaveFilesFeb04_2009.

```
agToEzFile=paste(myPathWavePath,"Data/agilentIDToEntrezGene.tab",sep="/")
agToEz=read.table(agToEzFile)
agToEz=as.matrix(agToEz)
```

The already normalized gene expression data is stored in the file Neuroblastoma_vsn_expr_stage1_4amp.tab. It should also be read in as a matrix.

```
exprFile=paste(myPathWavePath,"Data/Neuroblastoma_vsn_expr_stage1_4amp.tab",sep="/")
x.org=read.table(exprFile,row.names=NULL)
x.org=as.matrix(x.org)
```

Two vectors are set, one containing the Agilent probe Ids, and one containing the corresponding Entrez Gene Ids.

```
agilent.probes=agToEz[,1]
entrez.IDs=as.numeric(agToEz[,2])
```

Extract the unique Agilent Ids of the gene expression data (stored in the first column of matrix **x.org**) .

```
agilent.annotation=x.org[,1]
all.agilentIDs=unique(agilent.annotation)
```

To perform a statistical analysis, the class of each sample needs to be known. For the analysis here, the classes are patients with tumors of stage 1 and patients with tumors of stage 4 MYCN amplified.

```
clFile=paste(myPathWavePath,"Data/Neuroblastoma_vsn_expr_stage1_4amp.class",sep="/")
cl=read.table(clFile)
```

From these classes, a factor, **y**, is build whose elements are the class type and class name corresponding to the columns of the gene expression data matrix.

```
y=NULL
names=colnames(x.org)[colnames(x.org)!="row.names"]
for(i in colnames(x.org)){
  y=c(y,as.vector(cl$V2)[i == cl$V1])
}
y=as.factor(y)
```

Now, the gene expression data is mapped onto the KEGG reactions. The matrix reac.entrez contains the reaction Ids and the genes that code for these reactions. For every reaction all genes mapping onto it are listed, if more than one gene maps, they are separated by "_".

```
reac.entrez=NULL
for(maps in names(keggXml$genes)){
  toDel=NULL
  #Check if more than one gene maps to a reaction
  duplicate.reactions=unique(names(keggXml$genes[[maps]])[duplicated(names(keggXml$genes[[maps]]))])
  for(i in duplicate.reactions){
    reac.entrez=rbind(reac.entrez,c(i,paste((keggXml$genes[[maps]][grep(i,names(keggXml$genes[[maps]]))],collapse="_")))
    toDel=c(toDel,grep(i,names(keggXml$genes[[maps]])))
  }
  if(is.null(toDel)){
    rest.reactions=names(keggXml$genes[[maps]])
  } else{
    rest.reactions=names(keggXml$genes[[maps]])[-toDel]
  }
  for(i in rest.reactions){
    reac.entrez=rbind(reac.entrez,c(i,(keggXml$genes[[maps]])[grep(i,names(keggXml$genes[[maps]]))]))
  }
}
index=order(reac.entrez[,1])
reac.entrez=reac.entrez[index,]
```

The expression values for every reaction are calculated. If more than one gene maps onto a reaction, the mean value of their gene expression values is taken. This results in a matrix **x** with the mapped expression data.

```
x=NULL
x.names=NULL
for(i in 1:nrow(reac.entrez)){
  reac.entrezIDs=unlist(strsplit(reac.entrez[i,2],"_"))
  #Get agilent IDs
  reac.agilent=NULL
  for(j in reac.entrezIDs){
    help=agilent.probes[grep(paste("^",j,"$",sep=""),entrez.IDs,perl=TRUE)]
    if(length(help)>0){
      reac.agilent=c(reac.agilent,help)
    }
  }
  #Bind expression data together
  rows=agilent.annotation %in% reac.agilent
  if(length(rows[rows])>0){
    help=x.org[rows,2:ncol(x.org)]
    #Convert values into numbers (were read in as characters)
    #If more than one gene maps on a reaction: calculate mean value
    if(is.null(dim(help))){
      help=as.numeric(help)
    } else{
      help=as.matrix(apply(help,2,as.numeric))
      help=apply(help,2,mean)
    }
    x=rbind(x,help)
    x.names=c(x.names,reac.entrez[i,1])
  }
}
rownames(x)=x.names
```

To build the adjacency matrices for every KEGG pathway, the method *pwAdjMatrices()* is used. From these adjacency matrices the optimization problems can be formulated using *pwLPFiles()*. Furthermore, the result of the optimally ordered grids is read in, as is done further below

```
adj=pwAdjMatrices(keggXml$reactions,keggXml$compounds)
```

Two possibilities exist to get the optimal ordered grids:
On the web page the result can be downloaded in an R-readable format:
Download the RData file optimalGridHSA.RData into the working directory and load it

```
load("optimalGridHSA.RData")
```

Or
Up-load the result of the optimization algorithm into R using method *pwOptGrids()*. The log files are stored in folder
OptLogFiles form the downloaded PathWaveFilesFeb04_2009.

```
lp.path=paste(myPathWavePath,"OptLogFiles/",sep="")
optimalGridHSA=pwOptGrids(lp.path,adj)
```

Reactions that are in **x** (matrix with mapped gene expression values) but not in **optimalGridHSA** will be removed.

```
all.reac=unlist(lapply(optimalGridHSA,function(entry){unique(as.vector(entry$M))[unique(as.vector(entry$M))!="0"]}))
all.reac=unique(all.reac)
toDel=which(is.na(match(rownames(x),all.reac)))
if(length(toDel)>0){
  x=x[-toDel,]
}
```

Before starting the analysis the gene expression values are converted into z-scores

```
x=t(apply(x,1,function(entry){(entry-mean(entry))/sd(entry)}))
```

It is given out how many reactions are in the optimal arranged grids and for how many expression values exist,

```
print(paste("KEGG reactions:",length(all.reac),sep=" "))
print(paste("reactions with expression values:",nrow(x),sep=" "))
```

Now the analysis is performed using method *pathWave()*. How to use it and more details about the possible parameters can be found on the help page of *PathWave* (by typing ?pathWave)

```
result=pathWave(x,y,optimalM=optimalGridHSA,pvalCutoff=0.01,genes=keggXml$genes)
```

To get an overview over the results, a table is created. This table corresponds to table 1 in Schramm et al. (submitted).

```
#get table 1.
pval=NULL
score=NULL
all=NULL
down=NULL
up=NULL

for(i in names(result)){
  pval=c(pval,result[[i]]$p.value)
  score=c(score,result[[i]]$score)
  all=c(all,length(grep(i,rownames(x))))
  up=c(up,length(result[[i]]$reaction.regulation[result[[i]]$reaction.regulation<0]))
  down=c(down,length(result[[i]]$reaction.regulation[result[[i]]$reaction.regulation>0]))
}
res=cbind(up+down,all,up,down,pval,score)
rownames(res)=names(result)

print(res)
```

Methods in package *PathWave*:

1. pathWave()

Description: Main analysis method of package *PathWave*. Performs an enrichment analysis on optimally arranged grids. Features are generated using a Haar wavelet transform.

Usage:

```
pathWave(x, y, optimalM,
         mTest = TRUE,
         mTestMethod = "Bonferroni",
         pvalCutoff = 0.01,
         genes=NULL,
         diffReac = 5,
         nperm = 10000,
         verbose = TRUE)
```

Arguments:

x:	Matrix of expression values. Names of row elements correspond to elements in optimalM. Columns are data samples.
y:	Class factor for x. Should consist of only two classes and length(y) should correspond to ncol(x).
optimalM:	List of optimal grids as returned by function *pwOptGrids()*. See details.
mTest:	Should the results be corrected for multiple testing (default TRUE)?
mTestMethod:	Method for multiple testing correction (default "Bonferroni") as defined by package multtest.
pvalCutoff:	Significance level (default 0.01).
genes:	List of genes for each reaction per pathway as returned by *pwKEGGxml()*. See details.
diffReac:	Pathways with how many differentially expressed reaction should be considered (default 5)?
nperm:	Number of permutations that should be used to estimate the underlying distribution (default 10,000)?
verbose:	Should the progress in permutations be printed after every 100 permutations?

Details:

x	Rownames(x) should correspond to the reactionIDs in optimalM as the values of x will be mapped onto the matrices. If for an entry in optimalM no entry in x can be found, it is set to 0.
optimalM	List of the optimal arranged grid for each pathway. The Matrix consists of "0" and the Reaction IDs. The structure of the list: optimalM$pathwayID$M.
genes	List Element "genes" from list returned by *pwKEGGXml()*. For every pathwayID a vector is stored with the geneIDs. The names of the vector are the reactionIDs. The reactionIDs can occur more than once, if more than one gene is mapped onto the reaction.

Value:

PathWave returns a list of class PathWave:

p.value	P-value of the enriched pathway.
score	Size of the feature with which the score was calculated. This is a measurement of the size of the significant pattern.
reactions	Reactions from which the significant features were calculated by a Haar wavelet transform.
reaction.p.value	The p-values of the reactions. Calculated by a t-test on x.
reaction.regulation	The regulation of the reactions: +1 up-regulated, -1 down-regulated, 0 not differentially regulated in class levels(y)[1].
feature.p.value	The p-values of the features.
feature	List of the significant features.

2. pwAdjMatrices()

Description: Builds a list of and -if wanted prints- adjacency matrices for a list of bipartite graphs.

Usage:

```
pwAdjMatrices(reactionList, compoundList,
              printMatrices = FALSE,
              matrixType = "adjacency")
```

Arguments:

reactionList:	List of reactions. See details.
compoundList:	List of compounds. See details.
printMatrices:	Built matrices are printed in the working directory (default FALSE). See details.
matrixType:	Printed matrices could be of type "adjacency" or "distance" (default "adjacency").

Details:

reactionList	List of reactions as returned by pwKEGGxml -> reactions. List of graph IDs. For every graph ID a vector of the reaction IDs is stored.
compoundList	List of compoundss as returned by pwKEGGxml -> compounds. List of graph IDs. For every graph ID various list of reaction IDs are stored. Every reaction ID list has a vector of compounds connected to the specific reaction.
printMatrices	File names for printed matrices are built by concatenating pathway ID with ending ".matrix". First two numbers in file are number of nodes and length of shortest path.

Value:

A list is returned:

pathway ID:	List of different pathways.
nNodes:	Integer, number of nodes (reactions) in graph.
iMax:	Integer, length of longest shortest path between all pairs of reactions in graph.
M:	Adjacency matrix for the pathway ID.
dist:	Distance matrix for the pathway ID.

3. pwKEGGxml()

Description: Reads and parses KEGG xml files. Extracts gene IDs, reaction IDs, compound IDs and internal KEGG IDs for every pathway.

Usage:

```
pwKEGGxml(url = "ftp://ftp.genome.jp/pub/kegg/xml/organisms/hsa/")
```

Arguments:

url:	Url or directory containing the xml files that should be parsed (default "ftp://ftp.genome.jp/pub/kegg/xml/organisms/hsa/"). See details.

Details:

url	Variable url can either be an url or a directory. The function searches for xml files in the given url. IMPORTANT: reaction IDs are extracted for each pathway. To avoid mismatching similar reaction IDs of different pathways, pathway ID and reaction ID are concatenated, e.g. hsa00251:R00256 .

Value:

A List is returned:

ids:	Internal KEGG IDs. Contains a list of pathway IDs. For every pathway ID a vector is stored with the internal KEGG IDs for every reaction ID.
genes:	Contains a list of pathway IDs. For every pathway ID a vector is stored with the entrez gene IDs for every reaction ID. Reaction IDs can occur more than once if more than one gene can be mapped on it.
reactions:	Contains a list of pathway IDs. For every pathway ID a vector is stored with reaction IDs (concatenated with the pathway ID).
compounds:	Contains a list of pathway IDs. For every pathway ID a list of the pathways reaction IDs is stored. These lists contain the compounds the specific reaction is connected to.

4. pwLPFiles()

Description: Generates files of inequalities to be solved by a standard LP solver. Output files are in CPLEX format.

Usage:

```
pwLPFiles(adjMatrix,
          strategy = "automatic",
          addTriangleInequalities = TRUE,
          addCliqueConstraints = TRUE,
          addSymmetryBreakingConstraints = TRUE,
          addStarInequalities = TRUE,
          add5CycleConstraints = TRUE,
          useDummyNodes = TRUE,
          addNeighbourStarConstraints = FALSE)
```

Arguments:

adjMatrix:	List of adjacency matrices as returned by function *pwAdjMatrices()*.
strategy:	File generating strategy. Choices are "automatic" (default),"manually","largesystem" and "basicsystem". See details.
addTriangleInequalities:	Boolean variable. Can be changed if strategy "manually" is chosen (default TRUE). See details.
addCliqueConstraints:	Boolean variable. Can be changed if strategy "manually" is chosen (default TRUE). See details.
addSymmetryBreakingConstraints:	Boolean variable. Can be changed if strategy "manually" is chosen (default TRUE). See details.
addStarInequalities:	Boolean variable. Can be changed if strategy "manually" is chosen (default TRUE). See details.
add5CycleConstraints:	Boolean variable. Can be changed if strategy "manually" is chosen (default TRUE). See details.
useDummyNodes:	Boolean variable. Can be changed if strategy "manually" is chosen (default TRUE). See details.
addNeighbourStarConstraints:	Boolean variable. Can be changed if strategy "manually" is chosen (default FALSE). See details.

Details:

The function searches for constraining symmetries in the graph to facilitate the optimisation problem.

strategy	The file generating strategy can be adapted to the size of the adjacency matrices. Choice "automatic" does this automatically, "largesystem" could be used for larger adjacency matrices. Strategy "basicsystem" derives only the simplest basic constraints that are necessary to solve the optimisation problem. With strategy "manually" the user can decide what inequalities should be considered for the optimisation problem.
addTriangleInequalities	Choice only valid if strategy "manually" is chosen. Searches for triangles in the graph. The sum of the manhattan distances for the three neighbouring nodes must be at least 4.
addCliqueConstraints	Choice only valid if strategy "manually" is chosen. Searches for cliques in the graph. A lower limit for the sum of the member distances can be set.
addSymmetryBreakingConstraints	Choice only valid if strategy "manually" is chosen. Searches for topological conformations.
addStarInequalities	Choice only valid if strategy "manually" is chosen. Builds inequalities with lower limits for hub like structures in the graph.
add5CycleConstraints	Choice only valid if strategy "manually" is chosen. Builds inequalities with lower limits for circles with 5 members in the graph.
useDummyNodes	Choice only valid if strategy "manually" is chosen. Dummy nodes are introduced, filling up the adjacency matrix. Strangely, this can speed up CPLEX.
addNeighbourStarConstraints	Choice only valid if strategy "manually" is chosen. Searches for special structures from which inequalities can be built.

The output files can be read into CPLEX or other LP solver. They should also be usable with GNUs GLPK.

Value:

For each input pathway a file -pathwayID.lp- is written out into the working directory.

5. pwOptGrids()

Description: Parses output files that were solved by CPLEX and generates the optimal ordered grids. The output files must follow the problem structure formulated by function *pwLPFiles()*.

Usage: pwOptGrids(path, adjMatrix, pattern = "\.lp\.log$")

Arguments:

path:	Path where the function should search for the output files.
adjMatrix:	List of adjacency matrices as returned by *pwAdjMatrices()* and with which the LP problem was derived (via *pwLPFiles()*).
pattern:	The specific pattern of the file names.

Details:

The function allows the easy parsing of the solutions returned by CPLEX. It searches for lines starting with "x" followed by "_" and numbers, e.g. "x_1_1_1" following the formulation of the optimisation problems by *pwLPFiles()*. The found file names are split via "." assuming that the pathway ID is the first string in the file name.

Value:

A list is returned:

pathway ID:	For every pathway ID a list element M is defined with the ordered grid.

Note:

This parser was written to parse CPLEX results. For using different LP solvers changing the code might be necessary.