HEMDAG Documentation

Marco Notaro, Max Schubach, Giorgio Valentini

Mar 28, 2021

HEMDAG package:

- implements several Hierarchical Ensemble Methods (HEMs) for Directed Acyclic Graphs (DAGs);
- reconciles flat predictions with the topology of the ontology;
- can enhance the predictions of virtually any flat learning methods by taking into account the hierarchical relationships between ontology classes;
- guarantees biologically meaningful predictions that always obey the *true-path-rule*, the biological and logical rule that governs the internal coherence of biomedical ontologies;
- is specifically designed for exploiting the hierarchical relationships of DAG-structured taxonomies, such as the Human Phenotype Ontology (HPO) or the Gene Ontology (GO), but can be safely applied to tree-structured taxonomies as well (e.g. FunCat), since trees are DAGs;
- scales nicely both in terms of the complexity of the taxonomy and in the cardinality of the examples;
- provides several utility functions to process and analyze graphs;
- provides several performance metrics to evaluate HEMs algorithms.

CHAPTER 1

Quickstart

This short *HowTo* guides you from downloading HEMDAG library, load it into your R environment and make first computations.

1.1 Installation

Please go to the *Installation* section and install HEMDAG by using one of the ways shown.

1.2 Load HEMDAG Library

Start R in your console using

```
$ R
```

then load the library by using

```
> library(HEMDAG)
```

1.3 First Classification – for the Impatient

HEMDAG encompasses in total 23 hierarchical ensemble methods. Below we show the *simple* call to all the hierarchical ensemble algorithms included in HEMDAG, bu using the pre-built datasets available in the HEMDAG for making predictions. For more details about datasets and methods have a look to section *Tutorial*.

A. Loading the pre-built dataset of HEMDAG

```
# load the DAG g
> data(graph);

# load the scores matrix S
> data(scores);

# load the annotation matrix L
> data(labels);

# compute the root node
> root <- root.node(g);
```

B. HTD-DAG: Hierarchical Top-Down for DAG

```
> S.htd  <- htd(S, g, root);
```

C. GPAV: Generalized Pool-Adjacent-Violators

```
> S.gpav <- gpav.over.examples(S, g, W=NULL);
```

D. TPR-DAG (True Path Rule for DAG) and all its 18 ensemble variants

```
> S.tprTF          <- tpr.dag(S, g, root, positive="children", bottomup="threshold.free
↪", topdown="htd");
> S.tprT           <- tpr.dag(S, g, root, positive="children", bottomup="threshold",␣
↪topdown="htd", t=0.5);
> S.tprW           <- tpr.dag(S, g, root, positive="children", bottomup="weighted.
↪threshold.free", topdown="htd", w=0.5);
> S.tprWT          <- tpr.dag(S, g, root, positive="children", bottomup="weighted.
↪threshold", topdown="htd", t=0.5, w=0.5);

> S.descensTF      <- tpr.dag(S, g, root, positive="descendants", bottomup="threshold.
↪free", topdown="htd");
> S.descensT       <- tpr.dag(S, g, root, positive="descendants", bottomup="threshold",
↪ topdown="htd", t=0.5);
> S.descensW       <- tpr.dag(S, g, root, positive="descendants", bottomup="weighted.
↪threshold.free", topdown="htd", w=0.5);
> S.descensWT      <- tpr.dag(S, g, root, positive="descendants", bottomup="weighted.
↪threshold", topdown="htd", t=0.5, w=05);
> S.descensTAU     <- tpr.dag(S, g, root, positive="descendants", bottomup="tau",␣
↪topdown="htd", t=0.5);

> S.isotprTF       <- tpr.dag(S, g, root, positive="children", bottomup="threshold.free
↪", topdown="gpav");
> S.isotprT        <- tpr.dag(S, g, root, positive="children", bottomup="threshold",␣
↪topdown="gpav", t=0.5);
> S.isotprW        <- tpr.dag(S, g, root, positive="children", bottomup="weighted.
↪threshold.free", topdown="gpav", w=0.5);
> S.isotprWT       <- tpr.dag(S, g, root, positive="children", bottomup="weighted.
↪threshold", topdown="gpav", t=0.5, w=0.5);

> S.isodescensTF   <- tpr.dag(S, g, root, positive="descendants", bottomup="threshold.
↪free", topdown="gpav");
> S.isodescensT    <- tpr.dag(S, g, root, positive="descendants", bottomup="threshold",
↪ topdown="gpav", t=0.5);
> S.isodescensW    <- tpr.dag(S, g, root, positive="descendants", bottomup="weighted.
↪threshold.free", topdown="gpav", w=0.5);
```

(continues on next page)

(continued from previous page)

```
> S.isodescensWT  <- tpr.dag(S, g, root, positive="descendants", bottomup="weighted.
↪threshold", topdown="gpav", t=0.5, w=0.5);
> S.isodescensTAU <- tpr.dag(S, g, root, positive="descendants", bottomup="tau",␣
↪topdown="gpav", t=0.5);
```

E. Obozisnki heuristic methods

```
> S.max <- obozinski.max(S,g,root);
> S.and <- obozinski.and(S,g,root);
> S.or  <- obozinski.or(S,g,root);
```

CHAPTER 2

Installation

HEMDAG is available on CRAN, through Bioconda and from source code. You can use one of the following ways for installing HEMDAG.

2.1 Installation via Conda

This is the recommended way to install HEMDAG for normal users because it will enable you to switch software versions easily. In addition R with all needed dependencies will be installed.

First, you have to install the Miniconda Python3 distribution. See here for installation instructions. Make sure to:

- install the Python3 version of Miniconda.
- answer yes to the question whether conda shall be put into your PATH.

Then, you can install HEMDAG with

```
$ conda install -c bioconda -c conda-forge r-hemdag
```

from the Bioconda channel.

2.2 Global Installation

You can directly install the library via R by typing in your terminal:

```
$ R -e 'install.packages("HEMDAG", repos="http://cran.us.r-project.org")'
```

Alternatively, you can install the HEMDAG library by typing in the R environment:

```
install.packages("HEMDAG");
```

Another possibility to install the development version of HEMDAG is by using the *devtools* package (link):

```
library(devtools);
install_github("marconotaro/hemdag");
```

2.3 Dependencies

To install or build HEMDAG the following dependencies are required:

- R (2.10)
- R dependencies
  - graph (bioconductor)
  - rbgl (bioconductor)
  - precrec
  - preprocessCore (bioconductor)
  - plyr
  - foreach
  - doParallel

Note: CRAN does not automatically install Bioconductor packages. To install them:

```
if(!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")

BiocManager::install("graph")
BiocManager::install("rbgl")
BiocManager::install("preprocessCore")
```

2.4 Installing from Source

Here we describe how to build HEMDAG from scratch.

2.4.1 Package from CRAN

On a Linux environment, download the package source from the CRAN repo and save it (for instance) in the folder `pippo`. Then type:

```
R CMD INSTALL pippo/HEMDAG_<pkg-version-number>.tar.gz
```

Note: Replace `<pkg-version-number>` with the version number of the downloaded HEMDAG package.

2.4.2 Direct Git Checkout

Note: You only need to install from source if you want to develop HEMDAG yourself.

Below, we will download the HEMDAG sources and build them in `~/hemdag`:

```
~ $ cd ~
~ $ git clone https://github.com/marconotaro/hemdag.git
```

2.4.3 Building

You can build HEMDAG by using:

```
R CMD build hemdag
```

This will generate the file `HEMDAG_<package-version-number>.tar.gz` and just install the package via:

```
R CMD INSTALL HEMDAG_<package-version-number>.tar.gz
```

CHAPTER 3

Usage of HEMDAG

For a detailed description of the functions available in the HEMDAG library please go to the CRAN page and have a look to the reference manual.

CHAPTER 4

Tutorial

In this tutorial we show a step-by-step application of HEMDAG to the hierarchical prediction of associations between human gene and abnormal phenotype. To this end we will use the small pre-built dataset available in the HEMDAG library. However, all the hierarchical ensemble methods encompassed in HEMDAG library can be run by using:

- any ontology listed in OBO foundry (link);
- any flat score matrix, achieved by using any flat classifier ranging from linear, to probabilistic methods, to neural networks, to gradient boosting and many others;
- any annotation matrix.

Of course, the number of terms among the graph, the flat score matrix and the annotation matrix must match.

Note: To run the experiments shown in this page, make sure you have installed the following requirements:

- HEMDAG >= 2.7.4
- R >= 3.4.4
- Ubuntu >= 16.04

4.1 Load the HEMDAG Library

To load the HEMDAG library, open the R environment and type:

```
library(HEMDAG);
```

4.2 Load the Flat Scores Matrix

In their more general form, the hierarchical ensemble methods adopt a two-step learning strategy:

- the first step consists in the flat learning of the ontology terms;
- the second step *reconciles* the flat predictions by considering the topology of the underlying ontology.

Consequently, the first *ingredient* that we need in a hierarchical ensemble classification is the flat score matrix. For the sake of simplicity, in the examples shown below we use the pre-built dataset available in the HEMDAG library. To load the flat score matrix, type in the the R environment:

```
data(scores);
```

With the above command we loaded the flat score matrix `S`, that is a named 100 X 23 matrix. Rows correspond to genes (Entrez GeneID) and columns to HPO terms/classes. The scores represent the likelihood that a given gene belongs to a given class: the higher the value, the higher the likelihood that a gene belongs to a given class. This flat score matrix was obtained by running the RANKS package (link).

4.2.1 Normalization

Since RANKS **returns a score and not a probability**, we must normalize the scores of the matrix `S` to make the flat scores comparable with the hierarchical ones. In case the flat classifier returns directly a probability there is no needed to normalize the flat score matrix, since the flat scores can be directly compared with the hierarchical ones.

HEMDAG allows to normalize the flat scores according to two different procedures:

1. **maxnorm**: Normalization in the sense of the maximum: the score of each class is normalized by dividing the score values for the maximum score of that class:

```
S.maxnorm <- scores.normalization(norm.type="maxnorm", S);
```

2. **qnorm**: Quantile normalization: quantile normalization of the *preprocessCore* package is used:

```
S.qnorm <- scores.normalization(norm.type="qnorm", S);
```

Be sure to install the *preprocessCore* package before running the above command. Yo can install it by conda (`conda install -c bioconda bioconductor-preprocesscore`) or by Bioconductor (link)

Note: For the sake of simplicity, in all the examples shown in section *Hierarchical Ensemble Methods*, the input flat score matrix was normalized according to the `maxnorm` normalization:

```
S.norm <- scores.normalization(norm.type="maxnorm", S);
```

4.3 Load the Graph

In order to know how the hierarchical structure of the HPO terms, we need to load the graph:

```
data(graph);
```

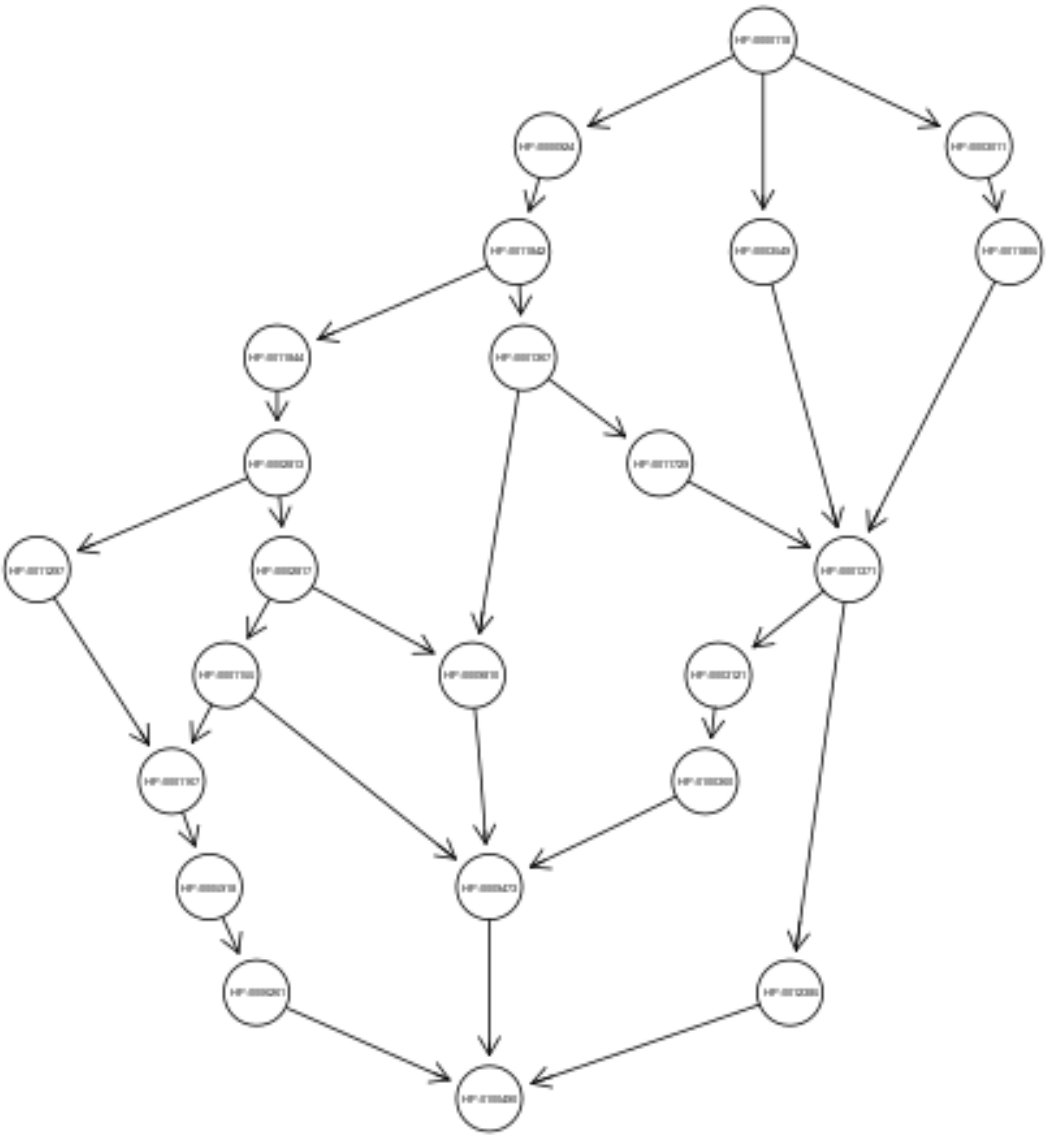
With the above command we loaded the graph `g`, an object of class `graphNEL`. The graph `g` has 23 nodes and 30 edges and represents the *ancestors view* of the HPO term `Camptodactyly of finger` (HP:0100490). Nodes of the graph `g` correspond to terms of the flat score matrix `S`.

4.3.1 Plot the Graph (optional)

Note: To plot the graph you need to install before the *Rgraphviz* package. Yo can install this library for example by conda (`conda install -c bioconda bioconductor-rgraphviz`) or by Bioconductor (link).

If you want to visualize the *ancestors view* of the term `HP:0100490`, just type:

```
library(Rgraphviz);
plot(g);
```



4.3.2 Utility Functions for Graphs (optional)

HEMDAG includes 33 utility functions (listed below) to process and analyze graphs as well as I/O functions to import a graph as object of class `graphNEL` or to export a graph as object of class `graphNEL` in a plain text file (in the classical tupla format). For more details on these functions, refer to the reference manual.

```
build.ancestors                          build.parents                          ␣
↪constraints.matrix
build.ancestors.bottom.up                build.parents.bottom.up                ␣
↪distances.from.leaves
build.ancestors.per.level                build.parents.top.down                 ␣
↪find.leaves
build.children                           build.parents.topological.sorting      ␣
↪graph.levels
build.children.bottom.up                 build.scores.matrix.from.list          ␣
↪lexicographical.topological.sort
build.children.top.down                  build.scores.matrix.from.tupla         ␣
↪read.graph
build.consistent.graph                   build.subgraph                         ␣
↪read.undirected.graph
build.descendants                        check.dag.integrity                    ␣
↪root.node
build.descendants.bottom.up              check.hierarchy                        ␣
↪tupla.matrix
build.descendants.per.level              check.hierarchy.single.sample          ␣
↪weighted.adjacency.matrix
build.edges.from.hpo.obo                 compute.flipped.graph                  ␣
↪write.graph
```

4.4 Hierarchical Ensemble Methods

First of all, we need to find the root node (i.e. node that is at the top-level of the hierarchy) of the HPO graph `g`. To do that just type:

```
root <- root.node(g);
```

in this way we store in the variable `root` the root node of the graph `g`.

Now, we are ready to run any ensemble algorithms implemented in the HEMDAG package.

4.4.1 HTD-DAG: Hierarchical Top Down for DAG

The HTD-DAG algorithm modifies the flat scores according to the hierarchy of a DAG G through a unique run across the nodes of the graph. For a given example x, the flat predictions $f(x) = \hat{y}$ are hierarchically corrected to $\bar{y}$, by per-level visiting the nodes of the DAG from top to bottom according to the following simple rule:

$$\bar{y}_i := \begin{cases} \hat{y}_i & \text{if} \quad i \in root(G) \\ \min_{j \in par(i)} \bar{y}_j & \text{if} \quad \min_{j \in par(i)} \bar{y}_j < \hat{y}_i \\ \hat{y}_i & \text{otherwise} \end{cases}$$

The node levels correspond to their maximum path length from the root. To call the HTD-DAG algorithm just type:

```
S.htd <- htd(S.norm, g, root);
```

Alternatively, we can call the `htd.vanilla` function (instead of `htd`), which it allows to normalize the flat score matrix `S` (according to **maxnorm** or **qnorm** normalization) *on the fly*:

run a normalization method (between **maxnorm** and **qnrom**) *on the fly*:

```
S.htd <- htd.vanilla(S, g, norm=TRUE, norm.type="maxnorm");
```

Note: In `htd.vanilla`, if `norm=FALSE` and `norm.type=NULL` the flat score matrix `S` is not normalized.

4.4.2 GPAV: Generalized Pool-Adjacent-Violators

Burdakov et al. in [1] proposed an approximate algorithm, named GPAV, to solve the *isotonic regression* (IR) or *monotonic regression* (MR) problem in its general case (i.e. partial order of the constraints). GPAV algorithm combines both low computational complexity (estimated to be $\mathcal{O}(|V|^2)$, where V is the number of nodes of the graph) and high accuracy. Formally, given a vector of observed values $\hat{y} \in R^n$, a strictly positive vector of weights $w \in R^n$ and a dag $G(V, E)$, GPAV finds the vector of fitted values $\bar{y} \in \mathbb{R}^n$ that solves the following convex quadratic program:

$$\begin{array}{ll} \min\limits_{\bar{y}} & \sum\limits_{i \in V} w_i(\bar{y}_i - \hat{y}_i)^2 \\ s.t. & \bar{y}_j \geq \bar{y}_i \quad \forall (i,j) \in E \end{array} \tag{4.1}$$

To call the GPAV algorithm just type:

```
S.gpav <- gpav.over.examples(S.norm, g, W=NULL);
```

It is worth noting that there is also a parallel version of the GPAV algorithm:

```
S.gpav <- gpav.parallel(S.norm, g, W=NULL, ncores=8);
```

Similarly to HTD-DAG also for GPAV, we can use the function `gpav.vanilla` (instead of `gpav.over.examples` or `gpav.parallel`) to normalize the flat score matrix `S` (according to **maxnorm** or **qnorm** normalization) *on the fly*:

```
S.gpav <- gpav.vanilla(S, g, W=NULL, parallel=TRUE, ncores=8, norm=TRUE, norm.type=
↪"maxnorm");
```

4.4.3 TPR-DAG: True Path Rule for DAG

TPR-DAG is a family of algorithms on the basis of the choice of the **bottom-up** step adopted for the selection of *positive* children. Indeed, in their more general form, the TPR-DAG algorithms adopt a two step learning strategy:

1. in the first step they compute a *per-level bottom-up* visit from leaves to root to propagate *positive* predictions across the hierarchy;
2. in the second step they compute a *per-level top-down* visit from root to leaves in order to assure the consistency of the predictions. In other word, the *HTD-DAG: Hierarchical Top Down for DAG* algorithm is applied.

Note: Levels (both in the first and second step) are defined in terms of the maximum path length from the root node. Please refer to [3] for further details.

The *vanilla* TPR-DAG adopts a per-level bottom-up traversal of the DAG to modify the flat predictions $\hat{y}_i$ according to the following formula:

$$\bar{y}_i := \frac{1}{1+|\phi_i|}(\hat{y}_i + \sum_{j \in \phi_i} \bar{y}_j)$$

where ϕ_i are the positive children of i (parameter `positive="children"`).

Different strategies to select the positive children ϕ_i can be applied:

1. **threshold-free** strategy (parameter `bottom="threshold.free"`): the positive nodes are those children that can increment the score of the node i, that is those nodes that achieve a score higher than that of their parents:

$$\phi_i := \{j \in child(i) | \bar{y}_j > \hat{y}_i\}$$

2. **threshold** strategy (parameter `bottom="threshold"`): the positive children are selected on the basis of a threshold that can be selected in two different ways:

   a) a unique threshold $\bar{t}$ is a priori selected for all nodes to determine the set of positives

$$\phi_i := \{j \in child(i) | \bar{y}_j > \bar{t}\}, \forall i \in V$$

   For instance if the predictions represent probabilities it could be meaningful set $\bar{t} = 0.5$.

   b) a threshold is selected to maximize some imbalance-aware performance metric $\mathcal{M}$ estimated on the training data, as for instance the Fmax or the AUPRC. In other words, the threshold is selected to maximize the measure $\mathcal{M}(j,t)$ on the training data for the term j with respect to the threshold t. The corresponding set of positives for each $i \in V$ is:

$$\phi_i := \{j \in child(i) | \bar{y}_j > t_j^*, t_j^* = \arg\max_t \mathcal{M}(j,t)\}$$

   Internal cross-validation is used to select t_j^* within a set of possible thresholds $t \in (0,1)$;

The weighted TPR-DAG version (parameter `bottom="weighted.threshold.free"`) can be designed by adding a weight $w \in [0,1]$ to balance the contribution of the parent node i and its positive children ϕ:

$$\bar{y}_i := w\hat{y}_i + \frac{(1-w)}{|\phi_i|} \sum_{j \in \phi_i} \bar{y}_j$$

If $w = 1$ no weight is attributed to the children and the TPR-DAG reduces to the HTD-DAG algorithm. If $w = 0$ only the predictors associated to the children nodes vote to predict node i. In the intermediate cases we attribute more importance to the predictor for the node i or to its children depending on the values of w.

By combining the weighted and the threshold variant, we design the *weighted-threshold* variant (parameter `bottom="weighted.threshold"`).

All the *vanilla* TPR-DAG variants use the HTD-DAG algorithm in the top-down step (parameter `topdown="htd"`) to provide ontology-based predictions (i.e. predictions that are coherent with the ontology structure):

```
S.tprTF <- tpr.dag(S.norm, g, root, positive="children", bottomup="threshold.free",␣
↪topdown="htd");
S.tprT  <- tpr.dag(S.norm, g, root, positive="children", bottomup="threshold",␣
↪topdown="htd", t=0.5);
S.tprW  <- tpr.dag(S.norm, g, root, positive="children", bottomup="weighted.threshold.
↪free", topdown="htd", w=0.5);
S.tprWT <- tpr.dag(S.norm, g, root, positive="children", bottomup="weighted.threshold
↪", topdown="htd", t=0.5, w=0.5);
```

4.4.4 DESCENS: Descendants Ensemble Classifier

As shown in [5] for tree-based hierarchies, the contribution of the descendants of a given node decays exponentially with their distance from the node itself and it is straightforward to see that this property also holds for DAG structured taxonomies. To overcame this limitation and in order to enhance the contribution of the most specific nodes to the overall decision of the ensemble we design the ensemble variant DESCENS. The novelty of DESCENS consists in strongly considering the contribution of all the descendants of each node instead of only that of its children (`positive="descendants"`). Therefore DESCENS predictions are more influenced by the information embedded in the leaves nodes, that are the classes containing the most informative and meaningful information from a biological and medical standpoint. DESCENS variants can be designed on the choice of the *positive* descendants Δ_i. The same strategies adopted for the choice of ϕ_i can be also adopted for the choice of Δ_i, simply by replacing ϕ_i with Δ_i and $child(i)$ with $desc(i)$ in the various formulas shown in *TPR-DAG: True Path Rule for DAG*. Furthermore, we designed a variant specific only for DESCENS, that we named DESCENS-τ (parameter `bottomup="tau"`). The DESCENS-τ variant balances the contribution between the *positives* children of a node i and that of its *positives* descendants excluding its children by adding a weight $\tau \in [0, 1]$:

$$\bar{y}_i := \frac{\tau}{1+|\phi_i|}(\hat{y}_i + \sum_{j \in \phi_i} \bar{y}_j) + \frac{1-\tau}{1+|\delta_i|}(\hat{y}_i + \sum_{j \in \delta_i} \bar{y}_j)$$

where ϕ_i are the *positive* children of i and $\delta_i = \Delta_i \setminus \phi_i$ the descendants of i without its children.

If $\tau = 1$ we consider only the contribution of the *positive* children of i; if $\tau = 0$ only the descendants that are not children contribute to the score, while for intermediate values of τ we can balance the contribution of ϕ_i and δ_i positive nodes.

All the DESCENS variants adopt in the second step the HTD-DAG algorithm to assure the consistency of the predictions:

```
S.descensTF  <- tpr.dag(S.norm, g, root, positive="descendants", bottomup="threshold.
↪free", topdown="htd");
S.descensT   <- tpr.dag(S.norm, g, root, positive="descendants", bottomup="threshold",
↪ topdown="htd", t=0.5);
S.descensW   <- tpr.dag(S.norm, g, root, positive="descendants", bottomup="weighted.
↪threshold.free", topdown="htd", w=0.5);
S.descensWT  <- tpr.dag(S.norm, g, root, positive="descendants", bottomup="weighted.
↪threshold", topdown="htd", t=0.5, w=05);
S.descensTAU <- tpr.dag(S.norm, g, root, positive="descendants", bottomup="tau",␣
↪topdown="htd", t=0.5);
```

4.4.5 ISO-TPR: Isotonic Regression for DAG

The ISO-TPR algorithms (parameter `positive="children"` and `topdown="gpav"`) considering the **positive children** in the bottom-up step and adopt GPAV (*GPAV: Generalized Pool-Adjacent-Violators*) instead of HTD-DAG (*HTD-DAG: Hierarchical Top Down for DAG*) in the consistency step. The most important feature of the ISO-TPR

algorithms is that they maintain the hierarchical constraints by construction by selecting the closest solution (in the least square sense) to the bottom-up predictions that obey the *True Path Rule*:

```
S.isotprTF <- tpr.dag(S.norm, g, root, positive="children", bottomup="threshold.free",
↪ topdown="gpav");
S.isotprT  <- tpr.dag(S.norm, g, root, positive="children", bottomup="threshold",␣
↪topdown="gpav", t=0.5);
S.isotprW  <- tpr.dag(S.norm, g, root, positive="children", bottomup="weighted.
↪threshold.free", topdown="gpav", w=0.5);
S.isotprWT <- tpr.dag(S.norm, g, root, positive="children", bottomup="weighted.
↪threshold", topdown="gpav", t=0.5, w=0.5);
```

4.4.6 ISO-DESCENS: Isotonic Regression with Descendants Ensemble Classifier

The ISO-DESCENS variants (parameter `positive="descendants"` and `topdown="gpav"`) considering the **positive descendants** instead of **positive children** in the bottom-up step and adopt GPAV (instead of the HTD-DAG algorithm) to guarantee the consistency of the predictions:

```
S.isodescensTF  <- tpr.dag(S.norm, g, root, positive="descendants", bottomup=
↪"threshold.free", topdown="gpav");
S.isodescensT   <- tpr.dag(S.norm, g, root, positive="descendants", bottomup=
↪"threshold", topdown="gpav", t=0.5);
S.isodescensW   <- tpr.dag(S.norm, g, root, positive="descendants", bottomup=
↪"weighted.threshold.free", topdown="gpav", w=0.5);
S.isodescensWT  <- tpr.dag(S.norm, g, root, positive="descendants", bottomup=
↪"weighted.threshold", topdown="gpav", t=0.5, w=0.5);
S.isodescensTAU <- tpr.dag(S.norm, g, root, positive="descendants", bottomup="tau",␣
↪topdown="gpav", t=0.5);
```

4.4.7 Obozinski Heuristic Methods

HEMDAG includes also the three heuristics ensemble methods (And, Max, Or) proposed in [4]:

1. **Max**: reports the largest logistic regression (LR) value of self and all descendants: $p_i = max_{j \in descendants(i)} \hat{p_j}$;
2. **And**: reports the product of LR values of all ancestors and self. This is equivalent to computing the probability that all ancestral terms are "on" assuming that, conditional on the data, all predictions are independent: $p_i = \prod_{j \in ancestors(i)} \hat{p_j}$;
3. **Or**: computes the probability that at least one of the descendant terms is "on" assuming again that, conditional on the data, all predictions are independent: $1 - p_i = \prod_{j \in descendants(i)} (1 - \hat{p_j})$;

To call Obozinski's heuristic methods, just type:

```
S.max <- obozinski.max(S.norm, g, root);
S.and <- obozinski.and(S.norm, g, root);
S.or  <- obozinski.or(S.norm, g, root);
```

Alternatively, the Obozinski's methods can be also called by properly setting the parameter `heuristic` of the function `obozinski.methods`:

```
S.max <- obozinski.methods(S, g, heuristic="max", norm=TRUE, norm.type="maxnorm");
S.and <- obozinski.methods(S, g, heuristic="and", norm=TRUE, norm.type="maxnorm");
S.or  <- obozinski.methods(S, g, heuristic="or",  norm=TRUE, norm.type="maxnorm");
```

4.5 Check Hierarchical Constraints

Predictions returned by a flat classifier **do not respect** the *True Path Rule* (since they neglect the structural information between different ontology terms), whereas the predictions returned by a hierarchical ensemble methods **always obey** the *True Path Rule*. According to this rule a *positive* instance for a class implies *positive* instance for all the ancestors of that class. We can easily check this fact by using the function `check.hierarchy`. Below (as an example) we check the consistency of the scores corrected according to the HTD-DAG strategy. Of course, all the scores matrices corrected with any hierarchical ensemble variants included in HEMDAG, respect the **True Path Rule**. We leave to the user the freedom to check the consistency of the scores matrix of the remaining 22 hierarchical ensemble variants encompassed in HEMDAG.

```
check.hierarchy(S, g, root)$status
"NOTOK"

check.hierarchy(S.htd, g, root)$status
"OK"
```

4.6 Performance Evaluation

To know the behavior of the hierarchical ensemble methods, the HEMDAG library provides both *term-centric* and *protein-centric* performance metrics:

- `AUPRC`: area under the precision-recall curve;
- `AUROC`: area under the ROC curve;
- `Fmax` : maximum hierarchical F-score [2];
- `PXR` : precision at different recall levels;

Note:

1. HEMDAG allows to compute all the aforementioned performance metrics either **one-shot** or **averaged** across k fold. Depending on the dataset size, the metrics `Fmax` and `PXR` could take a while to finish. Please refer to HEMDAG reference manual for further information about the input arguments of these functions.
2. For computing the *term-centric* metrics (`AUROC`, `AUPRC` and `PXR`), HEMDAG makes use of the R package *precrec* (link).

4.6.1 Load the Annotation Matrix

To compare the hierarchical ensemble methods against the flat approach, we need to load the annotation matrix:

```
data(labels);
```

With the above command we loaded the annotations table `L`, that is a named `100 X 23` matrix. Rows correspond to genes (`Entrez GeneID`) and columns to HPO terms/classes. `L[i, j] = 1` means that the gene `i` belong to class `j`, `L[i, j] = 0` means that the gene `i` does not belong to class `j`.

4.6.2 Flat vs Hierarchical

Before computing performance metrics we should remove the root node from the annotation matrix, the flat score matrix and the hierarchical scores matrix. Indeed, it does not make sense to take into account the predictions of the root node, since it is a *fake* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). In R this can be accomplished in one line of code.

```
## remove root node from annotation matrix
if(root %in% colnames(L))
    L <- L[,-which(colnames(L)==root)];

## remove root node from the normalized flat score matrix
if(root %in% colnames(S.norm))
    S.norm <- S.norm[,-which(colnames(S.norm)==root)];

## remove root node from hierarchical scores matrix (eg S.htd)
if(root %in% colnames(S.htd))
    S.htd <- S.htd[,-which(colnames(S.htd)==root)];
```

Now we can compare the flat approach RANKS versus the HTD-DAG strategy, by averaging (for instance) the performance across 3 folds:

```
## RANKS
prc.flat  <- auprc.single.over.classes(L, S.norm, folds=3, seed=23);
auc.flat  <- auroc.single.over.classes(L, S.norm, folds=3, seed=23);
pxr.flat  <- precision.at.given.recall.levels.over.classes(L, S.norm, recall.
↪levels=seq(from=0.1, to=1, by=0.1), folds=3, seed=23);
fmax.flat <- compute.fmax(L, S.norm, n.round=3, verbose=FALSE, b.per.example=TRUE,␣
↪folds=3, seed=23);

## HTD-DAG
prc.htd  <- auprc.single.over.classes(L, S.htd, folds=3, seed=23);
auc.htd  <- auroc.single.over.classes(L, S.htd, folds=3, seed=23);
pxr.htd  <- precision.at.given.recall.levels.over.classes(L, S.htd, recall.
↪levels=seq(from=0.1, to=1, by=0.1), folds=3, seed=23);
fmax.htd <- compute.fmax(L, S.htd, n.round=3, verbose=FALSE, b.per.example=TRUE,␣
↪folds=3, seed=23);
```

By looking at the results, it easy to see that the HTD-DAG outperforms the flat classifier RANKS:

```
## AUC performance: RANKS VS HTD-DAG
auc.flat$average
0.8297
auc.htd$average
0.8336

## PRC performance: RANKS VS HTD-DAG
prc.flat$average
0.4333
prc.htd$average
0.4627

## Fmax performance: RANKS VS HTD-DAG
fmax.flat$average
    P      R      S      F     avF      A      T
0.5042 0.8639 0.4485 0.6368 0.5269 0.6612 0.5720
fmax.htd$average
```

(continues on next page)

(continued from previous page)

```
    P      R      S      F    avF      A      T
0.5576 0.7745 0.6519 0.6484 0.5617 0.7521 0.6487

## PXR: RANKS VS HTD-DAG
pxr.flat$average
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.5821 0.5821 0.5821 0.5531 0.5531 0.4483 0.4388 0.4388 0.4388 0.4388
pxr.htd$average
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.6218 0.6218 0.6218 0.5941 0.5941 0.4798 0.4668 0.4668 0.4668 0.4668
```

Note: HTD-DAG is the simplest ensemble approach among those available. HTD-DAG strategy makes flat scores consistent with the hierarchy by propagating from top to bottom the negative predictions. Hence, in the worst case might happen that the predictions at leaves nodes are all negatives. Other ensemble algorithms, such as GPAV and TPR-DAG (and variants) should lead to better improvements.

4.7 Tuning of Hyper-Parameter(s)

14 out of 18 of the TPR-DAG hierarchical algorithms are parametric. Instead of use a fixed threshold (as done in *TPR-DAG: True Path Rule for DAG*), we can tune the hyper-parameter(s) of the parametric variants through the function `tpr.dag.cv`. The hyper-parameter(s) can be maximize on the basis of `AUPRC` (parameter `metric="prc"`) or `Fmax` (parameter `metric="fmax"`). Below, as an example, we maximize the threshold of the parametric variant `isotprT` on the basis of `AUPRC` metric.

```
threshold <- seq(0.1, 0.9, 0.1);

S.isotprT <- tpr.dag.cv(S, g, ann=L, norm=TRUE, norm.type="maxnorm", positive=
↪"children",
                        bottomup="threshold", topdown="gpav", W=NULL, parallel=FALSE,
                        ncores=1, threshold=threshold, weight=0, kk=3, seed=23,
                        metric="auprc", n.round=NULL);

## stdout
maxnorm normalization: done
training fold:  1   top auprc avg found:    0.4743567   best threshold: 0.1
training fold:  1   top auprc avg found:    0.4883769   best threshold: 0.5
training fold:  2   top auprc avg found:    0.2249245   best threshold: 0.1
training fold:  2   top auprc avg found:    0.2274687   best threshold: 0.3
training fold:  2   top auprc avg found:    0.2469059   best threshold: 0.4
training fold:  3   top auprc avg found:    0.8167777   best threshold: 0.1
training fold:  3   top auprc avg found:    0.8264204   best threshold: 0.3
training fold:  3   top auprc avg found:    0.8329289   best threshold: 0.7
tpr-dag correction done
```

Evaluating `isotprT` by computing *term*- and *protein*- centric performance (always averaging the performance across 3 folds), it easy to see how this ensemble variant outperform both the flat classifier RANKS and the hierarchical algorithm HTD-DAG:

```
## remove root node before computing performance
if(root %in% colnames(S.isotprT))
    S.isotprT <- S.isotprT[,-which(colnames(S.isotprT)==root)];
```

(continues on next page)

(continued from previous page)

```
prc.isotprT  <- auprc.single.over.classes(L, S.isotprT, folds=3, seed=23);
auc.isotprT  <- auroc.single.over.classes(L, S.isotprT, folds=3, seed=23);
pxr.isotprT  <- precision.at.given.recall.levels.over.classes(L, S.isotprT, recall.
↪levels=seq(from=0.1, to=1, by=0.1), folds=3, seed=23);
fmax.isotprT <- compute.fmax(L, S.isotprT, n.round=3, verbose=FALSE, b.per.
↪example=TRUE, folds=3, seed=23);

## AUC performance: RANKS VS HTD-DAG vs isotprT
auc.flat$average
0.8297
auc.htd$average
0.8336
auc.isotprT$average
0.8446

## PRC performance: RANKS VS HTD-DAG vs isotprT
prc.flat$average
0.4333
prc.htd$average
0.4627
prc.isotprT$average
0.5346

## Fmax performance: RANKS VS HTD-DAG vs isotprT
fmax.flat$average
    P      R      S      F    avF      A      T
0.5042 0.8639 0.4485 0.6368 0.5269 0.6612 0.5720
fmax.htd$average
    P      R      S      F    avF      A      T
0.5576 0.7745 0.6519 0.6484 0.5617 0.7521 0.6487
fmax.isotprT$average
    P      R      S      F    avF      A      T
0.5896 0.8306 0.5283 0.6896 0.6106 0.7066 0.6340

## PXR: RANKS VS HTD-DAG vs isotprT
pxr.flat$average
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.5821 0.5821 0.5821 0.5531 0.5531 0.4483 0.4388 0.4388 0.4388 0.4388
pxr.htd$average
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.6218 0.6218 0.6218 0.5941 0.5941 0.4798 0.4668 0.4668 0.4668 0.4668
pxr.isotprT$average
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.6848 0.6848 0.6848 0.6697 0.6697 0.5417 0.5027 0.5027 0.5027 0.5027
```

By properly setting the parameters `positive`, `bottomup` and `topdown` of the function `tpr.dag.cv`, it is easy to make experiments with all the 18 TPR-DAG ensemble variants. For further details on the other input arguments of the function `tpr.dag.cv`, please refer to the reference manual.

Note: Note that tuning the hyper-parameter(s) of the ensemble variants on the basis of `Fmax` might involve high running time (due to the nature itself of the `Fmax` metric).

4.8 Hold-out Functions

For all the hierarchical ensemble algorithms encompassed in the HEMDAG library there is also a corresponding hold-out version. The hold-out functions respect to the *vanilla* ones, require in input a vector of integer numbers corresponding to the indexes of the elements (rows) of the scores matrix `S` to be used in the test set (parameter `testIndex`). The hold-out ensemble functions included in HEMDAG are:

- `htd.holdout`;
- `gpav.holdout`;
- `tpr.dag.holdout`;
- `obozinski.holdout`;

For the sake of space we do not show here experiments by using the hold-out version of the hierarchical functions. Please refer to the reference manual, for further details on these functions.

References

CHAPTER 5

HEMDAG programmatic call and evaluation

Here we explain how to apply the ensemble algorithms of the HEMDAG family in both cross-validated and hold-out experiments.

HEMDAG can in principle boost the predictions of any flat learning method by reconciling the flat predictions with the topology of the underlying ontology. Hence, to run HEMDAG we need the following *ingredients*:

1) the label matrix `M` representing the protein annotations to functional terms;
2) the graph `g` representing the hierarchy of the functional terms;
3) the flat score matrix `S` representing a score or a probability that a gene/protein belongs to a given functional term;

To build the graph, the label matrix and the protein-protein interaction network you can use this pipeline. Instead, to obtain the flat score matrix you can use the shogun library or the caret package or any other software returning a score or a probability that a protein belongs to a functional term.

Note: HEMDAG is built upon flat predictions. HEMDAG corrects all the violations of the hierarchical relationships between ontology terms.

Note: To run the experiments shown below, make sure you have installed the following requirements:

- HEMDAG >= 2.7.4
- R >= 4.0.4
- Ubuntu >= 16.04

5.1 HEMDAG Call Script

To call any hierarchical ensemble algorithm of the HEMDAG family on either a time-lapse hold-out or a cross-validated dataset you can execute the following script:

```
#!/usr/bin/Rscript

## load library
library(HEMDAG);
suppressPackageStartupMessages(library(graph)); ## silence biocgenerics mask messages...
library(optparse);

## command line arguments
## for a detailed description, please see the manual: https://cran.r-project.org/web/packages/HEMDAG/HEMDAG.pdf
optionList <- list(
    make_option(c("-o", "--organism"), type="character", default="7227_drome",
        help="organism name in the form <taxon>_<name> (def. 7227_drome)"),
    make_option(c("-d", "--domain"), type="character", default="mf",
        help="go domain. It can be: bp, mf or cc (def. mf)"),
    make_option(c("-e", "--exptype"), type="character", default="ho",
        help="type of dataset on which run HEMDAG. It can be: ho (hold-out) or cv (cross-validated) -- def. ho"),
    make_option(c("-f", "--flat"), type="character", default="svmlinear",
        help="flat classifier"),
    make_option(c("-p", "--positive"), type="character", default="descendants",
        help="positive nodes selection. It can be: children or descendants.
                Skip this parameter if only topdown strategy is applied (def. descendants)"),
    make_option(c("-b", "--bottomup"), type="character", default="tau",
        help="bottomup strategy. It can be: none, threshold.free, threshold, weighted.threshold.free, weighted.threshold or tau.
                If none only topdown strategy is applied (def. tau)"),
    make_option(c("-t", "--topdown"), type="character", default="gpav",
        help="topdown strategy. It can be: htd or gpav (def. gpav)"),
    make_option(c("-c", "--threshold"), type="character", default="seq(from=0.1, to=0.9, by=0.1)",
        help="threshold for the choice of positive nodes.
                It can be a fixed value or an array of values (def. seq(from=0.1, to=0.9, by=0.1))"),
    make_option(c("-w", "--weight"), type="character", default="0",
        help="weight for the choice of positive nodes. It can be a fixed value or an array of values (def. 0)"),
    make_option(c("-m", "--metric"), type="character", default="auprc",
        help="performance metric on which maximize the parametric ensemble algorithms. It can be: auprc or fmax (def. auprc)"),
    make_option(c("-r", "--round"), type="integer", default="3",
        help="number of rounding digits to be applied for choosing the best Fmax. To be used only if metric is set to fmax (def. 3)"),
    make_option(c("-s", "--seed"), type="integer", default="23",
        help="seed for the random generator to create folds (def. 23)"),
    make_option(c("-k", "--fold"), type="integer", default="5",
        help="number of folds for the cross validation (def. 5)"),
    make_option(c("-l", "--parallel"), type="logical", default=FALSE, action="store_true",
        help="should the sequential or parallel version of gpav be run?
                If flag -p is 'on' the gpav parallel version is run. NB: only gpav can be run in parallel (def. FALSE)"),
    make_option(c("-n", "--cores"), type="integer", default="1",
        help="number of cores to use for the parallel execution of gpav (def. 1)"),
    make_option(c("-z", "--norm"), type="logical", default=FALSE, action="store_true",
```

(continues on next page)

(continued from previous page)

```
        help="should the flat score matrix be normalized? If flag -p is 'on' the input flat scores is normalized (def. FALSE)"),
    make_option(c("-y", "--normtype"), type="character", default="none",
        help="type of normalization. It can be maxnorm or qnorm (def. none)")
);

optParser <- OptionParser(option_list=optionList);
opt <- parse_args(optParser);

prefix    <- opt$organism;
organism  <- strsplit(prefix,"_")[[1]][2];
exptype   <- opt$exptype;
flat      <- opt$flat;
positive  <- opt$positive;
bottomup  <- opt$bottomup;
topdown   <- opt$topdown;
domain    <- opt$domain;
threshold <- eval(parse(text=opt$threshold));
weight    <- eval(parse(text=opt$weight));
metric    <- opt$metric;
round     <- opt$round;
seed      <- opt$seed;
kk        <- opt$fold;
parallel  <- opt$parallel;
cores     <- opt$cores;
norm      <- opt$norm;
normtype  <- opt$normtype;
if(normtype == "none")
    normtype <- NULL;

## hemdag algorithm to be displayed in output file name -> 18 iso/tpr-dag ensemble combinations + gpav + htd (tot 20 hemdag family)
if(positive=="children" && bottomup=="threshold.free" && topdown=="htd")
    hemdag.name <- "tprTF";
if(positive=="children" && bottomup=="threshold" && topdown=="htd")
    hemdag.name <- "tprT";
if(positive=="children" && bottomup=="weighted.threshold.free" && topdown=="htd")
    hemdag.name <- "tprW";
if(positive=="children" && bottomup=="weighted.threshold" && topdown=="htd")
    hemdag.name <- "tprwt";
if(positive=="descendants" && bottomup=="threshold.free" && topdown=="htd")
    hemdag.name <- "descensTF";
if(positive=="descendants" && bottomup=="threshold" && topdown=="htd")
    hemdag.name <- "descensT";
if(positive=="descendants" && bottomup=="weighted.threshold.free" && topdown=="htd")
    hemdag.name <- "descensW";
if(positive=="descendants" && bottomup=="weighted.threshold" && topdown=="htd")
    hemdag.name <- "descensWT";
if(positive=="descendants" && bottomup=="tau" && topdown=="htd")
    hemdag.name <- "descensTAU";
if(positive=="children" && bottomup=="threshold.free" && topdown=="gpav")
    hemdag.name <- "isotprTF";
if(positive=="children" && bottomup=="threshold" && topdown=="gpav")
    hemdag.name <- "isotprT";
if(positive=="children" && bottomup=="weighted.threshold.free" && topdown=="gpav")
    hemdag.name <- "isotprW";
if(positive=="children" && bottomup=="weighted.threshold" && topdown=="gpav")
```

(continues on next page)

(continued from previous page)

```
    hemdag.name <- "isotprWT";
if(positive=="descendants" && bottomup=="threshold.free" && topdown=="gpav")
    hemdag.name <- "isodescensTF";
if(positive=="descendants" && bottomup=="threshold" && topdown=="gpav")
    hemdag.name <- "isodescensT";
if(positive=="descendants" && bottomup=="weighted.threshold.free" && topdown=="gpav")
    hemdag.name <- "isodescensW";
if(positive=="descendants" && bottomup=="weighted.threshold" && topdown=="gpav")
    hemdag.name <- "isodescensWT";
if(positive=="descendants" && bottomup=="tau" && topdown=="gpav")
    hemdag.name <- "isodescensTAU";
if(bottomup=="none" && topdown=="gpav")
    hemdag.name <- "gpav";
if(bottomup=="none" && topdown=="htd")
    hemdag.name <- "htd";

## I/O directories
data.dir <- paste0("../data/", exptype, "/");
res.dir  <- paste0("../res/",  exptype, "/");
if(!dir.exists(res.dir)){dir.create(res.dir, recursive=TRUE);}

## flat/ann/dag/testIndex files
files <- list.files(data.dir);
flat.file <- files[grep(paste0(organism, ".*", domain, ".scores.*", flat), files)];
ann.file  <- files[grep(paste0(domain,".ann"), files)];
dag.file  <- files[grep(paste0(domain,".dag"), files)];
if(exptype == "ho"){
    idx.file  <- files[grep(paste0(domain,".testindex"), files)];
    if(length(idx.file)==0)
        stop("no index file found\n");
}

## check if flat|ann|dag exists
if(length(flat.file)==0 || length(ann.file)==0 || length(dag.file)==0)
    stop("no flat|ann|dag file found\n");

## load data
S <- get(load(paste0(data.dir, flat.file)));
g <- get(load(paste0(data.dir, dag.file)));
ann <- get(load(paste0(data.dir, ann.file)));
if(exptype == "ho")
    testIndex <- get(load(paste0(data.dir, idx.file)));

## shrink graph g to terms of matrix S -- if number of nodes between g and S mismatch
root <- root.node(g);
nd <- colnames(S);
class.check <- ncol(S) != graph::numNodes(g);
if(class.check){
    root.check <- root %in% colnames(S);
    if(!root.check)
        nd <- c(root, nd);
    g <- build.subgraph(nd, g, edgemode="directed");
    ann <- ann[, colnames(S)];
}

## address case when (iso)descensW is called with a fixed value of w to enter in the␣
↪right branch
```

(continues on next page)

(continued from previous page)

```
if(bottomup=="weighted.threshold.free" && length(weight)==1)
    threshold <- 0;

## elapsed time
start.elapsed <- proc.time();

## HEMDAG calling
if(exptype == "ho"){
    if(bottomup=="none"){
        if(topdown=="gpav"){
            S.hier <- gpav.holdout(S=S, g=g, testIndex=testIndex, W=NULL, parallel=parallel,
                ncores=cores, norm=norm, norm.type=normtype);
        }else{
            S.hier <- htd.holdout(S=S, g=g, testIndex=testIndex, norm=norm, norm.type=normtype);
        }
    }else{ ## branch to call HEMDAG by tuning the parameters
        if(length(threshold)>1 || length(weight)>1){
            S.hier <- tpr.dag.holdout(S, g, ann=ann, testIndex=testIndex, norm=norm, norm.type=normtype,
                positive=positive, bottomup=bottomup, topdown=topdown, W=NULL, parallel=parallel, ncores=cores,
                threshold=threshold, weight=weight, kk=kk, seed=seed, metric=metric, n.round=round);
        }else{ ## branch to call HEMDAG with fixed the parameters
            ## add root node if it does not exist
            if(!(root %in% colnames(S))){
                max.score <- max(S);
                z <- rep(max.score,nrow(S));
                S <- cbind(z,S);
                colnames(S)[1] <- root;
            }
            ## normalization
            if(norm){
                S <- scores.normalization(norm.type=normtype, S);
                cat(normtype, "normalization done\n");
            }
            ## shrink S to test indexes
            S.test <- S[testIndex,];
            ## degenerate case when test set has just one row/example
            if(!is.matrix(S.test)){
                test.sample <- rownames(S)[testIndex];
                S.test <- matrix(S.test, ncol=length(S.test), dimnames=list(test.sample, names(S.test)));
            }
            ## tpr-dag correction
            S.hier <- tpr.dag(S.test, g, root=root, positive=positive, bottomup=bottomup, topdown=topdown,
                t=threshold, w=weight, W=NULL, parallel=parallel, ncores=cores);
            ## print chosen parameters
            if(bottomup=="weighted.threshold.free"){
                cat("fixed weight:", weight, "\n");
            }else if(bottomup=="weighted.threshold"){
                cat("fixed weight:", weight, "fixed threshold:", threshold, "\n");
            }else{
                cat("fixed threshold:", threshold, "\n");
```

(continues on next page)

(continued from previous page)

```
            }
            cat("tpr-dag correction done\n");
        }
    }
}else{
    if(bottomup=="none"){
        if(topdown=="gpav"){
            S.hier <- gpav.vanilla(S=S, g=g, W=NULL, parallel=parallel, ncores=cores, norm=norm, norm.type=normtype);
        }else{
            S.hier <- htd.vanilla(S=S, g=g, norm=norm, norm.type=normtype);
        }
    }else{  ## branch to call HEMDAG by tuning the parameters
        if(length(threshold)>1 || length(weight)>1){
            S.hier <- tpr.dag.cv(S, g, ann=ann, norm=norm, norm.type=normtype, positive=positive, bottomup=bottomup,
                topdown=topdown, W=NULL, parallel=parallel, ncores=cores, threshold=threshold, weight=weight,
                kk=kk, seed=seed, metric=metric, n.round=round);
        }else{  ## branch to call HEMDAG with fixed parameters
            ## add root node if it does not exist
            if(!(root %in% colnames(S))){
                max.score <- max(S);
                z <- rep(max.score,nrow(S));
                S <- cbind(z,S);
                colnames(S)[1] <- root;
            }
            ## normalization
            if(norm){
                S <- scores.normalization(norm.type=normtype, S);
                cat(normtype, "normalization done\n");
            }
            S.hier <- tpr.dag(S, g, root=root, positive=positive, bottomup=bottomup, topdown=topdown,
                t=threshold, w=weight, W=NULL, parallel=parallel, ncores=cores);
            ## print chosen parameters
            if(bottomup=="weighted.threshold.free"){
                cat("weight: ", weight, "\n");
            }else if(bottomup=="weighted.threshold"){
                cat("weight: ", weight, "threshold: ", threshold, "\n");
            }else{
                cat("threshold: ", threshold, "\n");
            }
            cat("tpr-dag correction done\n");
        }
    }
}

stop.elapsed <- proc.time() - start.elapsed;
timing.s <- stop.elapsed["elapsed"];
timing.m <- round(timing.s/(60),4);
timing.h <- round(timing.m/(60),4);
cat(hemdag.name, "running time:", timing.s["elapsed"], "(seconds)", "|", timing.m["elapsed"], "(minutes)", "|" , timing.h["elapsed"], "(hours)", "\n\n");

## store results
## outname
```

(continues on next page)

(continued from previous page)

```
fname <- unlist(strsplit(flat.file, split="[.,_]"));
outname <- paste0(fname[-((length(fname)-1):length(fname))], collapse="_");
if(norm==TRUE && !(is.null(normtype)))
    outname <- paste0(outname,"_",normtype);
if(exptype == "ho"){
    save(S.hier, file=paste0(res.dir, outname, "_", hemdag.name, "_holdout.rda"), compress=TRUE);
}else{
    save(S.hier, file=paste0(res.dir, outname, "_", hemdag.name, ".rda"), compress=TRUE);
}
```

You can download the script as follow:

```
mkdir -p ~/hemdag/script/
cd ~/hemdag/script/
wget -nc https://raw.githubusercontent.com/marconotaro/hemdag/master/docs/playground/script/hemdag-call.R
```

Before executing the script be sure to have correctly installed the latest version of the HEMDAG package (and all its dependencies – see *Installation*) and the package optparse.

Note:

1. The output hierarchical score matrix of the called HEMDAG algorithm (whose name is saved in the output *.rda* file name) is stored in the folder `~/hemdag/res/(ho|cv)` depending on whether you chose to execute HEMDAG on either hold-out (ho) or cross-validated (cv) datasets. The HEMDAG elapsed time is printed on the shell.
2. By default, if no inputs parameters are specified in `hemdag-call.R`, the script executes the `isodescensTAU` algorithm on the hold-out dataset by tuning the parameter `tau` on the basis of AUPRC.
3. The tuning of the hyper-parameters can take from few minutes up to few hours depending on the size of the dataset and on the adopted evaluation metric (Fmax is slower than AUPRC).

5.1.1 Arguments Explanation

For the usage of the script, type in the shell under the `~/hemdag/script/` folder:

```
Rscript hemdag-call.R -h
```

For a detailed description of the input arguments *positive, bottomup, topdown, threshold, weight, metric, round, seed, fold, parallel, cores, norm, normtype*, please refer to the description of the input variables of the functions `(gpav|htd|tpr.dag).(holdout|cv)` in the HEMDAG reference manual.

Parametric-free arguments

To call a parametric-free HEMDAG algorithm the main required arguments are:

- `-b (--bottomup) none`
- `-t (--topdown) gpav|htd`

Note: GPAV can also be run in parallel simply by using the flag `-l (--parallel)` and by setting the number of cores `-n (--cores)`.

Parametric arguments

To call a parametric HEMDAG algorithm the main required arguments are:

- `-b (--bottomup) threshold.free|threshold|weighted.threshold.free|weighted.threshold|tau`
- `-t (--topdown) gpav|htd`
- `-c (--cut-off) 0.5|"seq(from=0.1, to=0.9, by=0.1)"`. Note the use of double quotes for the range of thresholds
- `-w (--weight) 0.5|"seq(from=0.1, to=0.9, by=0.1)"`. Note the use of double quotes for the range of thresholds

If a range of thresholds (or weights) is selected, the hyper-parameters are tuned on the basis of imbalance-aware performance metrics estimated on the training data – `-m (--metric) auprc|fmax`. By default the number of folds `-k (--fold)` is set to 5 and the seed `-s (--seed)` for the random generator is set to 23. Furthermore, if `-m (--metric) fmax` the parameter `-r (--round)` can be used to select the number of rounding digits to be applied for choosing the best Fmax (by default is set to 3).

Additional arguments

The following arguments are dataset-specific:

- `-o (--organism)` specifies the organism name (in the form <taxon>_<name>);
- `-d (--domain)` specifies the GO domain: bp (biological process), mf (molecular function), cc(cellular component);
- `-e (--exptype)` specifies the type of dataset where running HEMDAG: ho (holdout) or cv (cross-validated);
- `-f (--flat)` specifies the name of the flat classifier. In case the flat learning method returns a score and not a probability, the flat score matrix must be normalized before running HEMDAG. On the contrary, if the flat classifier already returns a probability there is no needed to normalize the flat score matrix, since the flat scores can be directly compared with the hierarchical ones. To normalize the flat score matrix we must "activate" the flag `-z (--norm)` (by default the flag `-z` is deactivate) and we need to choose a type of normalization (`-y (--normtype) maxnorm|qnorm`). The name of the chosen normalization is stored in the *rda* output file name.

5.2 Time-lapse hold-out experiments

Here, to show how to use HEMDAG in time-lapse hold-out experiments, we use a pre-built dataset of the organism *Drosophila melanogaster* (DROME) and, for simplicity, we consider the annotations of the GO domain molecular function (MF). To build the dataset we used the annotations of an old GO release (December 2017) as training set and the annotations of a more recent GO release (June 2020) as test set. The graph and the annotation matrix was built by adopting the pipeline. The flat score matrix was obtained by using the R interface of the machine learning library LiblineaR with the default parameter settings. For further details on the dataset, please refer to *HEMDAG: a*

family of modular and scalable hierarchical ensemble methods to improve Gene Ontology term prediction (submitted to Bioinformatics).

5.2.1 Download Data

All the required *.rda* files can be downloaded by using the following commands, by exploiting the beauty and power of the non-greedy positive lookbehind regex :

```
mkdir -p ~/hemdag/data/ho/
cd ~/hemdag/data/ho/
curl -Ss https://github.com/marconotaro/hemdag/tree/master/docs/playground/data/ho | grep -oP '(?<=href=").*?(?=\">)' | grep '.rda$' | perl -pe 's/blob\///' | perl -pe 's/^/https\:\/\/raw.githubusercontent.com/' | wget -nc -i -
```

With the command above, we download the following datasets:

- `7227_drome_go_mf_ann_20dec17_16jun20.rda`: the annotation matrix;
- `7227_drome_go_mf_dag_20dec17_16jun20.rda`: the graph;
- `7227_drome_go_mf_testindex_20dec17_16jun20.rda`: the indexes of the examples of the test set;
- `7227_drome_go_mf_scores_svmlinear_holdout.rda`: the flat score matrix;

5.2.2 Programmatic Call

Below we show some examples of how to call HEMDAG in time-lapse hold-out experiments, but we leave the user the freedom to experiment with any another ensemble algorithm of the HEMDAG family.

Note:

1. the `hemdag-call.R` script must be called in `~/hemdag/script/`;
2. for the examples shown below, the tuning of hyper-parameters takes few minutes;
3. the output HEMDAG score matrices are stored in `~/hemdag/res/ho/`.

`GPAV` call (parallel version):

```
Rscript hemdag-call.R -o 7227_drome -d mf -e ho -f svmlinear -b none -t gpav -l -n 12
```

`isotprTF` call:

```
Rscript hemdag-call.R -o 7227_drome -d mf -e ho -f svmlinear -p children -b threshold.free -t gpav -l -n 12
```

`isotprW` call:

```
Rscript hemdag-call.R -o 7227_drome -d mf -e ho -f svmlinear -p children -b weighted.threshold.free -t gpav -w "seq(from=0.1, to=0.9, by=0.1)" -m auprc -s 23 -k 5 -l -n 12
```

`isodescensTF` call:

```
Rscript hemdag-call.R -o 7227_drome -d mf -e ho -f svmlinear -p descendants -b threshold.free -t gpav -l -n 12
```

`isodescensW` call:

```
Rscript hemdag-call.R -o 7227_drome -d mf -e ho -f svmlinear -p descendants -b␣
↪weighted.threshold.free -t gpav -w "seq(from=0.1, to=0.9, by=0.1)" -m auprc -s 23 -
↪k 5 -l -n 12
```

`isodescensTAU` call:

```
Rscript hemdag-call.R -o 7227_drome -d mf -e ho -f svmlinear -p descendants -b tau -t␣
↪gpav -c "seq(from=0.1, to=0.9, by=0.1)" -m auprc -s 23 -k 5 -l -n 12
```

5.2.3 Check Hierarchical Constraints

All the HEMDAG score matrices respect the hierarchical constraints imposed by the underlying ontology. The script below checks that all the 6 HEMDAG matrices obtained with the commands shown above, do not violate the between-term relationships in the GO MF hierarchy. For further details refer to *Check Hierarchical Constraints*.

```
#!/usr/bin/Rscript

suppressPackageStartupMessages(library(HEMDAG));
library(optparse);

optionList <- list(
    make_option(c("-o", "--organism"), type="character", default="7227_drome",
        help="organism name in the form <taxon>_<name> (def 7227_drome)"),
    make_option(c("-d", "--domain"),  type="character", default="mf",
        help="gene ontology domain -- bp, mf, cc (def. mf)"),
     make_option(c("-e", "--exptype"), type="character", default="ho",
        help="type of dataset on which run HEMDAG. It can be: ho (hold-out) or cv␣
↪(cross-validated) -- def. ho"),
    make_option(c("-f", "--flat"), type="character", default="svmlinear",
        help="flat classifier (def. svmlinear)"),
    make_option(c("-a", "--algorithm"), type="character", default="isodescensTAU",
        help="hierarchical correction algorithm (def. isodescensTAU)")
);

optParser <- OptionParser(option_list=optionList);
opt <- parse_args(optParser);

## setting input argument
prefix    <- opt$organism;
taxon     <- strsplit(prefix,"_")[[1]][1];
organism  <- strsplit(prefix,"_")[[1]][2];
domain    <- opt$domain;
exptype   <- opt$exptype;
flat      <- opt$flat;
algorithm <- opt$algorithm;

## I/O
data.dir <- paste0("../data/", exptype, "/");
res.dir  <- paste0("../res/",  exptype, "/");
data.files <- list.files(data.dir);
res.files  <- list.files(res.dir);
if(!dir.exists(res.dir)){dir.create(res.dir, recursive=TRUE);}

## load data
```

(continues on next page)

(continued from previous page)

```
dag.file  <- data.files[grep(paste0(domain,".dag"), data.files)];
hier.file <- res.files[grep(paste0(organism, ".*", domain, ".scores.*", flat, ".",
↪algorithm), res.files)];

## check if flat|ann|dag exists
if(length(dag.file)==0 || length(hier.file)==0)
    stop("no dag|hier file found\n");

## check constraints violation
g <- get(load(paste0(data.dir, dag.file)));
S.hier <- get(load(paste0(res.dir, hier.file)));
root <- root.node(g);
g <- build.subgraph(colnames(S.hier), g);
check <- check.hierarchy(S.hier, g, root);
if(check$status=="OK"){
    cat(taxon, organism, domain, paste0(flat,'+',algorithm), "check passed :)","\n");
}else{
    cat(taxon, organism, domain, paste0(flat,'+',algorithm), "check failed :(","\n");
}
```

To download and use the hierarchical constraints script:

```
## download
mkdir -p ~/hemdag/script/
cd ~/hemdag/script/
wget -nc https://raw.githubusercontent.com/marconotaro/hemdag/master/docs/playground/
↪script/hemdag-checker.R

## call
Rscript hemdag-checker.R -o 7227_drome -d mf -e cv -f ranger -a isodescensTAU
```

You can call `hemdag-checker.R` by looping through more hierarchical ensemble methods:

```
algotihms=("gpav" "isotprTF" "isotprW" "isodescensTF" "isodescensW" "isodescensTAU")

for ((i=0; i<${#algotihms[@]}; i++)); do
    Rscript hemdag-checker.R -o 7227_drome -d mf -e ho -f svmlinear -a ${algotihms[
↪$i]}
done;

## example of stdout
7227 drome mf ranger+gpav check passed :)
7227 drome mf ranger+isotprTF check passed :)
7227 drome mf ranger+isotprW check passed :)
7227 drome mf ranger+isodescensTF check passed :)
7227 drome mf ranger+isodescensW check passed :)
7227 drome mf ranger+isodescensTAU check passed :)
```

Of course, you can loop `hemdag-checker.R` also through the arguments `-o, -d, -e, -f` according with the values of your interest.

5.2.4 Evaluation

To evaluate the generalization performance of HEDMAG in the time-lapse hold-out experiments performed above, you can use the *term-centric* and/or *protein-centric* evaluation metrics provided by the HEMDAG package itself. For

further details on the implemented performance metric refer to section *Performance Evaluation* of the HEMDAG tutorial.

```
#!/usr/bin/Rscript

start.time <- proc.time();

library(HEMDAG);
library(optparse);

optionList <- list(
    make_option(c("-o", "--organism"), type="character", default="7227_drome",
        help="organism name in the form <taxon>_<name> (def 7227_drome)"),
    make_option(c("-d", "--domain"),  type="character", default="mf",
        help="gene ontology domain -- bp, mf, cc (def. mf)"),
     make_option(c("-e", "--exptype"), type="character", default="ho",
        help="type of dataset on which run HEMDAG. It can be: ho (hold-out) or cv
↪(cross-validated) -- def. ho"),
    make_option(c("-f", "--flat"), type="character", default="svmlinear",
        help="flat classifier (def. svmlinear)"),
    make_option(c("-a", "--algorithm"), type="character", default="isodescensTAU",
        help="hierarchical correction algorithm (def. isodescensTAU)")
);

optParser <- OptionParser(option_list=optionList);
opt <- parse_args(optParser);

## setting input argument
prefix    <- opt$organism;
taxon     <- strsplit(prefix,"_")[[1]][1];
organism  <- strsplit(prefix,"_")[[1]][2];
domain    <- opt$domain;
exptype   <- opt$exptype;
flat      <- opt$flat;
algorithm <- opt$algorithm;

## I/O
data.dir <- paste0("../data/", exptype, "/");
res.dir  <- paste0("../res/",  exptype, "/");
if(!dir.exists(res.dir)){dir.create(res.dir, recursive=TRUE);}

## flat/ann/dag/testIndex file
files <- list.files(data.dir);
flat.file <- files[grep(paste0(organism, ".*", domain, ".scores.*", flat), files)];
ann.file  <- files[grep(paste0(domain,".ann"), files)];
dag.file  <- files[grep(paste0(domain,".dag"), files)];
## check if flat|ann|dag exists
if(length(flat.file)==0 || length(ann.file)==0 || length(dag.file)==0)
    stop("no flat|ann|dag file found\n");

if(exptype == "ho"){
    idx.file  <- files[grep(paste0(domain,".testindex"), files)];
    if(length(idx.file)==0)
        stop("no idx file found\n");
}

## hier file
files <- list.files(res.dir);
```

(continues on next page)

(continued from previous page)

```
hier.file <- files[grep(paste0(organism, ".*", domain, ".scores.*", flat, ".", algorithm), files)];

## load data
S <- get(load(paste0(data.dir, flat.file)));
S.hier <- get(load(paste0(res.dir, hier.file)));
g <- get(load(paste0(data.dir, dag.file)));
ann <- get(load(paste0(data.dir, ann.file)));
if(exptype == "ho")
    testIndex <- get(load(paste0(data.dir, idx.file)));

## if number of nodes between g and S mismatch -> shrink graph g to terms of matrix S
## eg. during flat learning you removed from S all those terms having less than N annotations
root <- root.node(g);
nd <- colnames(S);
class.check <- ncol(S) != graph::numNodes(g);
if(class.check){
    root.check <- root %in% colnames(S);
    if(!root.check)
        nd <- c(root, nd);
    g <- build.subgraph(nd, g, edgemode="directed");
    ann <- ann[, colnames(S)];
}

## remove root node S and S.hier score matrix (if any)
root <- root.node(g);
if(root %in% colnames(S)){
    S <- S[,-which(colnames(S)==root)];
    cat("root node removed from flat score matrix\n");
}
if(root %in% colnames(S.hier)){
    S.hier <- S.hier[,-which(colnames(S.hier)==root)];
    cat("root node removed from hierarchical score matrix\n");
}
## remove root node from annotation matrix (if any)
if(root %in% colnames(ann)){
    ann <- ann[,-which(colnames(ann)==root)];
    cat("root node removed from annotation matrix\n");
}

## shrink S to testIndex
if(exptype == "ho"){
    S <- S[testIndex, ];
    ann <- ann[testIndex, colnames(S)];
}

## compute flat perf
auc.flat  <- auroc.single.over.classes(target=ann, predicted=S, folds=NULL, seed=NULL);
prc.flat  <- auprc.single.over.classes(target=ann, predicted=S, folds=NULL, seed=NULL);
fmax.flat <- compute.fmax(target=ann, predicted=S, n.round=3, b.per.example=TRUE, folds=NULL, seed=NULL, verbose=FALSE);
pxr.flat  <- precision.at.given.recall.levels.over.classes(target=ann, predicted=S, folds=NULL, seed=NULL, recall.levels=seq(from=0.1, to=1, by=0.1));
cat(taxon, organism, domain, flat, algorithm, "flat performance done\n");
```

(continues on next page)

(continued from previous page)

```

## compute hier perf
auc.hier  <- auroc.single.over.classes(target=ann, predicted=S.hier, folds=NULL,␣
↪seed=NULL);
prc.hier  <- auprc.single.over.classes(target=ann, predicted=S.hier, folds=NULL,␣
↪seed=NULL);
fmax.hier <- compute.fmax(target=ann, predicted=S.hier, n.round=3, b.per.example=TRUE,
↪ folds=NULL, seed=NULL, verbose=FALSE);
pxr.hier  <- precision.at.given.recall.levels.over.classes(target=ann, predicted=S.
↪hier, folds=NULL, seed=NULL, recall.levels=seq(from=0.1, to=1, by=0.1));
cat(taxon, organism, domain, flat, algorithm, "hierarchical performance done\n");

## storing
outname <- gsub("scores", "perfmeas", hier.file)
save(auc.flat, prc.flat, fmax.flat, pxr.flat, auc.hier, prc.hier, fmax.hier, pxr.hier,
↪ file=paste0(res.dir, outname), compress=TRUE);

## timing
timing.s <- proc.time() - start.time;
timing.m <- round(timing.s/(60),4);
timing.h <- round(timing.m/(60),4);
cat("\n");
cat("elapsed time:", timing.s["elapsed"], "(seconds)", "|", timing.m["elapsed"],
↪"(minutes)", "|" , timing.h["elapsed"], "(hours)", "\n\n");
```

To download and use the HEMDAG evaluation script:

```
## download
mkdir -p ~/hemdag/script/
cd ~/hemdag/script/
wget -nc https://raw.githubusercontent.com/marconotaro/hemdag/master/docs/playground/
↪script/hemdag-eval.R

## call
Rscript hemdag-eval.R -o 7227_drome -d mf -e ho -f svmlinear -a gpav
```

Chunk evaluation (optional)

The above R call evaluates the performance of an HEMDAG algorithm just on a single dataset. Since HEMDAG can be virtually applied on top of any flat classifier, the Perl script below generates "chunks" of HEMDAG evaluation calls.

Note: The parameter regulating the number of chunk is `$m`, set by default to 12. Increase (resp. decrease) this value to rise (reduce) the number of HEMDAG evaluation calls to be executed in parallel.

```
#!/usr/bin/perl

use strict;
use warnings;
use Getopt::Long 'HelpMessage';

GetOptions(
    "org=s"   => \(my @orgs=""),
```

(continues on next page)

(continued from previous page)

```
    "flat=s" => \(my @flats=""),
    "alg=s"  => \(my @algs=""),
    "dmn=s"  => \(my @onts=""),
    'exp=s'  => \(my $exp='ho'),
    'chk=i'  => \(my $chk='12'),
    'help'   => sub {HelpMessage(0)},
) or HelpMessage(1);

@orgs=  split(/,/,join(',',@orgs));
@flats= split(/,/,join(',',@flats));
@algs=  split(/,/,join(',',@algs));
@onts=  split(/,/,join(',',@onts));

print "#!/bin/sh\n\n";
print "tot_start=\$(date +%s)\n\n";

my $k=0;  ## cpu number on which binding a task
foreach my $org (@orgs){
    foreach my $flat (@flats){
        foreach my $alg (@algs){
            foreach my $ont (@onts){
                $k++;
                my $cpu= $k-1;
                print "taskset -c $cpu Rscript hemdag-eval.R -o $org -d $ont -e $exp -f $flat -a $alg > $org"."_go_"."$ont"."_"."$flat"."_"."$alg"."_perfmeas.out 2> /dev/null &\n";
                if($k % $chk ==0){
                    print "\n";
                    print "wait\n";
                    print "\n";
                    $k=0;
                }
            }
        }
    }
}

print "\n";
print "tot_end=\$(date +%s)\n";
print "tot_elapsed_s=\$((tot_end-tot_start))\n";
print "tot_elapsed_m=\$((tot_elapsed_s/60))\n";
print "tot_elapsed_h=\$((tot_elapsed_m/60))\n";
print "printf \"grand total elapsed time:\t\$((tot_elapsed_s)) SECONDS\t\$((tot_elapsed_m)) MINUTES\t\$((tot_elapsed_h)) HOURS\"\n";
print "echo\n\n";

print "echo \"compute performance done\"\n\n";

__END__

=pod

=head1 NAME

call-hemdag-chunk - generate HEMDAG evaluation calls chunks

=head1 SYNOPSIS
```

(continues on next page)

(continued from previous page)

```
63
64      --org,-o    organism list. The list can have one or more elements. The elements␣
    ↪must be comma separated (7227_drome,9031_chick,..)
65      --flt,-f    flat classifier list. The list can have one or more elements. The␣
    ↪elements must be comma separated (svmlinear,ranger,..)
66      --alg,-a    hierarchical correction algorithm list. The list can have one or more␣
    ↪elements. The elements must be comma separated (gpav,isodescensTF,isodescensTAU,..)
67      --dmn,-d    gene ontology domain list. The list can have one or more elements.␣
    ↪The elements must be comma separated (bp,mf,cc)
68      --exp,-e    type of dataset on which evaluate HEMDAG. It can be: ho or cv (def.␣
    ↪ho)
69      --chk,-c    number of parallel evaluations to be computed before going to the␣
    ↪next block (def 12)
70      --help,-h   print this help
71
72  =cut
```

To download and use the Perl script that generates "chunks" of HEMDAG evaluation calls:

```
## download the perl script
mkdir -p ~/hemdag/script/
cd ~/hemdag/script/
wget -nc https://raw.githubusercontent.com/marconotaro/hemdag/master/docs/playground/
↪script/hemdag-chunk.pl

## generate chunk evaluation calls
perl hemdag-chunk.pl -o 7227_drome -d mf -f svmlinear -a gpav,isotprTF,isotprW,
↪isodescensTF,isodescensW,isodescensTAU -e ho -c 6 > hemdag-ho-eval.sh

## evaluate HEMDAG in chunks
bash hemdag-ho-eval.sh > out &
```

You can generate the calls that you wish, simply by extending the arguments of the arrays *@orgs, @flats, @algs, @onts* with the wanted values. For instance, by setting `-d bp,mf,cc`, the Perl script `hemdag-chunk.pl` returns 2 chunks of evaluation calls, the first made of 12 calls and the second one of 6 calls:

```
## call
perl hemdag-chunk.pl -o 7227_drome -d bp,mf,cc -f svmlinear -a gpav,isotprTF,isotprW,
↪isodescensTF,isodescensW,isodescensTAU -e ho -c 12

## stdout

#!/bin/sh

tot_start=$(date +%s)

taskset -c 0 Rscript hemdag-eval.R -o 7227_drome -d bp -e ho -f svmlinear -a gpav >␣
↪7227_drome_go_bp_svmlinear_gpav_perfmeas.out 2> /dev/null &
taskset -c 1 Rscript hemdag-eval.R -o 7227_drome -d mf -e ho -f svmlinear -a gpav >␣
↪7227_drome_go_mf_svmlinear_gpav_perfmeas.out 2> /dev/null &
taskset -c 2 Rscript hemdag-eval.R -o 7227_drome -d cc -e ho -f svmlinear -a gpav >␣
↪7227_drome_go_cc_svmlinear_gpav_perfmeas.out 2> /dev/null &
taskset -c 3 Rscript hemdag-eval.R -o 7227_drome -d bp -e ho -f svmlinear -a isotprTF␣
↪> 7227_drome_go_bp_svmlinear_isotprTF_perfmeas.out 2> /dev/null &
taskset -c 4 Rscript hemdag-eval.R -o 7227_drome -d mf -e ho -f svmlinear -a isotprTF␣
↪> 7227_drome_go_mf_svmlinear_isotprTF_perfmeas.out 2> /dev/null &
```

(continues on next page)

(continued from previous page)

```
taskset -c 5 Rscript hemdag-eval.R -o 7227_drome -d cc -e ho -f svmlinear -a isotprTF␣
↪> 7227_drome_go_cc_svmlinear_isotprTF_perfmeas.out 2> /dev/null &
taskset -c 6 Rscript hemdag-eval.R -o 7227_drome -d bp -e ho -f svmlinear -a isotprW >
↪ 7227_drome_go_bp_svmlinear_isotprW_perfmeas.out 2> /dev/null &
taskset -c 7 Rscript hemdag-eval.R -o 7227_drome -d mf -e ho -f svmlinear -a isotprW >
↪ 7227_drome_go_mf_svmlinear_isotprW_perfmeas.out 2> /dev/null &
taskset -c 8 Rscript hemdag-eval.R -o 7227_drome -d cc -e ho -f svmlinear -a isotprW >
↪ 7227_drome_go_cc_svmlinear_isotprW_perfmeas.out 2> /dev/null &
taskset -c 9 Rscript hemdag-eval.R -o 7227_drome -d bp -e ho -f svmlinear -a␣
↪isodescensTF > 7227_drome_go_bp_svmlinear_isodescensTF_perfmeas.out 2> /dev/null &
taskset -c 10 Rscript hemdag-eval.R -o 7227_drome -d mf -e ho -f svmlinear -a␣
↪isodescensTF > 7227_drome_go_mf_svmlinear_isodescensTF_perfmeas.out 2> /dev/null &
taskset -c 11 Rscript hemdag-eval.R -o 7227_drome -d cc -e ho -f svmlinear -a␣
↪isodescensTF > 7227_drome_go_cc_svmlinear_isodescensTF_perfmeas.out 2> /dev/null &

wait

taskset -c 0 Rscript hemdag-eval.R -o 7227_drome -d bp -e ho -f svmlinear -a␣
↪isodescensW > 7227_drome_go_bp_svmlinear_isodescensW_perfmeas.out 2> /dev/null &
taskset -c 1 Rscript hemdag-eval.R -o 7227_drome -d mf -e ho -f svmlinear -a␣
↪isodescensW > 7227_drome_go_mf_svmlinear_isodescensW_perfmeas.out 2> /dev/null &
taskset -c 2 Rscript hemdag-eval.R -o 7227_drome -d cc -e ho -f svmlinear -a␣
↪isodescensW > 7227_drome_go_cc_svmlinear_isodescensW_perfmeas.out 2> /dev/null &
taskset -c 3 Rscript hemdag-eval.R -o 7227_drome -d bp -e ho -f svmlinear -a␣
↪isodescensTAU > 7227_drome_go_bp_svmlinear_isodescensTAU_perfmeas.out 2> /dev/null &
taskset -c 4 Rscript hemdag-eval.R -o 7227_drome -d mf -e ho -f svmlinear -a␣
↪isodescensTAU > 7227_drome_go_mf_svmlinear_isodescensTAU_perfmeas.out 2> /dev/null &
taskset -c 5 Rscript hemdag-eval.R -o 7227_drome -d cc -e ho -f svmlinear -a␣
↪isodescensTAU > 7227_drome_go_cc_svmlinear_isodescensTAU_perfmeas.out 2> /dev/null &

tot_end=$(date +%s)
tot_elapsed_s=$((tot_end-tot_start))
tot_elapsed_m=$((tot_elapsed_s/60))
tot_elapsed_h=$((tot_elapsed_m/60))
printf "grand total elapsed time:   $((tot_elapsed_s)) SECONDS  $((tot_elapsed_m))␣
↪MINUTES  $((tot_elapsed_h)) HOURS"
echo

echo "compute performance done"
```

5.3 Cross-validated experiments

Here, to show how to use HEMDAG in cross-validated experiments, we use a pre-built dataset of the organism *Drosophila melanogaster* (DROME) that covers the annotations of the GO domain molecular function (MF). The graph and protein-GO term associations belong to the GO release of December 2017. The graph and the annotation matrix was built by adopting the following pipeline. The flat score matrix was obtained by using the random forest as flat learning method (model *ranger* in the R library caret package with the default parameter settings). For further details on the dataset, please refer to *HEMDAG: a family of modular and scalable hierarchical ensemble methods to improve Gene Ontology term prediction (submitted to Bioinformatics)*.

5.3.1 Download Data

All the required *.rda* files can be downloaded by using the following commands:

```
mkdir -p ~/hemdag/data/cv/
cd ~/hemdag/data/cv/
curl -Ss https://github.com/marconotaro/hemdag/tree/master/docs/playground/data/cv | grep -oP '(?<=href=").*?(?=\">)' | grep '.rda$' | perl -pe 's/blob\///' | perl -pe 's/^/https\:\/\/raw.githubusercontent.com/' | wget -nc -i -
```

Note: Note the change of the last directory from `ho/` to `cv/` compared to the data downloaded in the *Time-lapse hold-out experiments*.

With the command above, we download the following datasets:

- `7227_drome_go_mf_ann_20dec17.rda`: the annotation matrix;
- `7227_drome_go_mf_dag_20dec17.rda`: the graph;
- `7227_drome_go_mf_scores_pearson_100_feature_ranger_5fcv.rda`: the flat score matrix;

5.3.2 Programmatic Call

To execute any HEMDAG algorithm on cross-validated datasets, basically you must replace `-e ho` with `-e cv` in the various calls shown in section *Programmatic Call* for the time-split hold-out experiments.

Note:

1. the `hemdag-call.R` script must be called in `~/hemdag/script/`;
2. for the examples shown below the tuning of hyper-parameters takes around one hour;
3. the flat classifier adopted for the cross-validated experiments performed here is not the *svm* (`-f svmlinear`), but the *random forest* (`-f ranger`);
4. the output HEMDAG score matrices are stored in `~/hemdag/res/cv/` (note the shift of the last directory);

For instance, to call the 6 HEMDAG on cross-validated datasets just type:

```
## GPAV
Rscript hemdag-call.R -o 7227_drome -d mf -e cv -f ranger -b none -t gpav -l -n 12

## isotprTF
Rscript hemdag-call.R -o 7227_drome -d mf -e cv -f ranger -p children -b threshold.free -t gpav -l -n 12

## isotprW
Rscript hemdag-call.R -o 7227_drome -d mf -e cv -f ranger -p children -b weighted.threshold.free -t gpav -w "seq(from=0.1, to=0.9, by=0.1)" -m auprc -s 23 -k 5 -l -n 12

## isodescensTF
Rscript hemdag-call.R -o 7227_drome -d mf -e cv -f ranger -p descendants -b threshold.free -t gpav -l -n 12

## isodescensW
Rscript hemdag-call.R -o 7227_drome -d mf -e cv -f ranger -p descendants -b weighted.threshold.free -t gpav -w "seq(from=0.1, to=0.9, by=0.1)" -m auprc -s 23 -k 5 -l -n 12
```

(continues on next page)

(continued from previous page)

```
## isodescensTAU
Rscript hemdag-call.R -o 7227_drome -d mf -e cv -f ranger -p descendants -b tau -t␣
↪gpav -c "seq(from=0.1, to=0.9, by=0.1)" -m auprc -s 23 -k 5 -l -n 12
```

5.3.3 Check Hierarchical Constraints

To check that HEMDAG does not violate hierarchical constraints imposed by the GO MF hierarchy in the cross-validated datasets obtained above, just replace replace `-e ho` with `-e cv` and `-f svmlinear` with `-f ranger` in the *Check Hierarchical Constraints* script:

```
algotihms=("gpav" "isotprTF" "isotprW" "isodescensTF" "isodescensW" "isodescensTAU")

for ((i=0; i<${#algotihms[@]}; i++)); do
    Rscript hemdag-checker.R -o 7227_drome -d mf -e cv -f ranger -a ${algotihms[$i]}
done;
```

5.3.4 Evaluation

To evaluate HEMDAG in the cross-validated experiments performed above, just replace replace `-e ho` with `-e cv` and `-f svmlinear` with `-f ranger` in the *Evaluation* script:

Note: the evaluation of the cross-validated dataset used here takes around 30 minutes – the slowest metric is `PXR`.

```
## single evaluation call
Rscript hemdag-eval.R -o 7227_drome -d mf -e cv -f ranger -a gpav

## generate chunk evaluation calls
perl hemdag-chunk.pl -o 7227_drome -d mf -f ranger -a gpav,isotprTF,isotprW,
↪isodescensTF,isodescensW,isodescensTAU -e cv -c 6 > hemdag-cv-eval.sh

## evaluate HEMDAG in chunks
bash hemdag-cv-eval.sh > out &
```

CHAPTER 6

Frequently Asked Questions

6.1 Where are the questions?

Right now, there are no frequently asked questions. Please contact the authors if you have questions.

CHAPTER 7

Cite HEMDAG

If you use HEMDAG package please cite the following references:

- Notaro M, Marco Frasca, Alessandro Petrini, Jessica Gliozzo, and Giorgio Valentini. HEMDAG: a family of modular and scalable hierarchical ensemble methods to improve gene ontology term prediction. Submitted to Bioinformatics.
- Marco Notaro, Max Schubach, Peter N. Robinson, and Giorgio Valentini. Prediction of Human Phenotype Ontology terms by means of hierarchical ensemble methods. *BMC Bioinformatics*, 18(1):449, December 2017. doi:10.1186/s12859-017-1854-y.

CHAPTER 8

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/marconotaro/hemdag/issues

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

8.1.2 Fix Bugs

Look through the Github issues for bugs. If you want to start working on a bug then please write short message on the issue tracker to prevent duplicate work.

8.1.3 Implement Features

Look through the Github issues for features. If you want to start working on an issue then please write short message on the issue tracker to prevent duplicate work.

8.1.4 Write Documentation

HEMDAG could always use more documentation, whether as part of the official HEMDAG docs, in docstrings, or even on the web in blog posts, articles, and such.

HEMDAG uses Sphinx for the user manual (that you are currently reading). See *doc_guidelines* on how the documentation reStructuredText is used. See *doc_setup* on creating a local setup for building the documentation.

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/marconotaro/hemdag/issues

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

8.2 Documentation Guidelines

For the documentation, please adhere to the following guidelines:

- Put each sentence on its own line, this makes tracking changes through Git SCM easier.
- Provide hyperlink targets, at least for the first two section levels.
- Use the section structure from below.

```
.. heading_1:

=========
Heading 1
=========


.. heading_2:

---------
Heading 2
---------


.. heading_3:

Heading 3
=========


.. heading_4:

Heading 4
---------


.. heading_5:
```

(continues on next page)

(continued from previous page)

```
Heading 5
~~~~~~~~~


.. heading_6:

Heading 6
:::::::::
```

8.3 Documentation Setup

For building the documentation, you have to install the Python program Sphinx. We use conda for that, see *Installation via Conda*

Use the following steps for installing Sphinx and the dependencies for building the HEMDAG documentation:

```
$ cd hemdag/docs
$ conda create --name sphinx --file environment.yml
$ source activate sphinx
```

Use the following for building the documentation. If you are not in the sphinx environment (e.g. you uses `source deactivate sphinx`) please activate the virtual environment using `source activate sphinx` Afterwards, you can always use `make html` for building.

```
(sphinx) $ cd hemdag/docs
(sphinx) $ make html  # rebuild for changed files only
(sphinx) $ make clean && make html  # force rebuild
```

8.4 Get Started!

Ready to contribute?

1. Fork the *hemdag* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/hemdag.git
   ```

3. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

4. When you're done making your changes, make sure that the build runs through.

   ```
   $ cd docs && make clean && make html
   ```

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

8.5 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated.
3. Describe your changes in the `NEWS.md` file.

CHAPTER 9

Authors

in alphabetical order

- Marco Notaro (maintainer)
- Max Schubach
- Giorgio Valentini

CHAPTER 10

History

10.1 HEMDAG 2.7.4

10.1.1 Changes

- remove extra input parameter `f.criterion` from `tpr.dag.cv`, `tpr.dag.holdout`, `find.best.f` and `compute.fmax`: type of F-measure used to select the best F-measure is always the harmonic mean between the average precision and recall (`f.criterion="F"`) and never the F-measure computed as average across examples (`f.criterion="avF"`);
- fix a minor bug in `tpr.dag.holdout`;
- add warning checks in `tpr.dag.cv` and `tpr.dag.holdout`;
- improve some test cases and manual;

10.2 HEMDAG 2.7.3

10.2.1 New Features

- add `build.scores.matrix.from.list`;
- add `build.scores.matrix.from.tupla`;
- add several test cases;

10.2.2 Changes

- streamline and lighten HEMDAG's hierarchical functions (namespace clearer and lighter);
- rename the following functions:
  - htd-dag:

    * Do.HTD -> htd.vanilla;
    * Do.HTD.holdout -> htd.holdout;
  - obozinski heuristic methods:
    * heuristic.max -> obozinski.max;
    * heuristic.and -> obozinski.and;
    * heuristic.or -> obozinski.or;
    * Do.heuristic.methods -> obozinski.methods;
    * Do.heuristic.methods.holdout -> obozinski.holdout;
  - gpav:
    * GPAV -> gpav;
    * GPAV.over.examples -> gpav.over.examples;
    * GPAV.parallel -> gpav.parallel;
    * Do.GPAV -> gpav.vanilla;
    * Do.GPAV.holdout -> gpav.holdout;
  - tpr-dag:
    * TPR.DAG -> tpr.dag;
    * Do.TPR.DAG -> tpr.dag.cv;
    * Do.TPR.DAG.holdout -> tpr.dag.holdout;
  - utility functions:
    * get.parents -> build.parents;
    * get.parents.top.down -> build.parents.top.down;
    * get.parents.bottom.up -> build.parents.bottom.up;
    * get.parents.topological.sorting -> build.parents.topological.sorting;
    * get.children.top.down -> build.children.top.down;
    * get.children.bottom.up -> build.children.bottom.up;
    * check.DAG.integrity -> check.dag.integrity;
    * do.subgraph -> build.subgraph;
    * do.submatrix -> build.submatrix;
    * do.stratified.cv.data.single.class -> stratified.cv.data.single.class;
    * do.stratified.cv.data.over.classes -> stratified.cv.data.over.classes;
    * do.unstratified.cv.data -> unstratified.cv.data;
    * do.edges.from.HPO.obo -> build.edges.from.hpo.obo;
  - performance metrics:
    * AUPRC.single.class -> auprc.single.class;
    * AUPRC.single.over.classes -> auprc.single.over.classes;

    * `AUROC.single.class` –> `auroc.single.class`;
    * `AUROC.single.over.classes` –> `auroc.single.over.classes`;
    * `compute.Fmeasure.multilabel` –> `compute.fmax`;
- remove the following functions (no more needed):
  - `Do.flat.scores.normalization`;
  - `Do.full.annotation.matrix`;
- improve manual;
- make HEMDAG's documentation clearer and less redundant;

10.3 HEMDAG 2.6.1

10.3.1 Changes

- fix `stringsAsFactors` issue – link;

10.4 HEMDAG 2.6.0

10.4.1 Changes

- fix NAMESPACE notes in CRAN checks;
- add link to the GitHub repository `obogaf::parser`;
- adjust link to read the docs;

10.5 HEMDAG 2.5.9

10.5.1 New Features

- add `build.consistent.graph`;

10.5.2 Changes

- add some warning checks in functions that compute performance metrics;
- improve some graph utility functions;
- improve manual;
- improve tutorial on read the docs – link;
- make namespace clearer;
- fix minor bugs;
- remove defunct functions;

10.6 HEMDAG 2.4.8

10.6.1 Changes

- fix a minor bug in `Do.GPAV.holdout`;
- improve package description;

10.7 HEMDAG 2.4.7

10.7.1 New Features

- fix degenerate case in `precision.at.all.recall.levels.single.class` (labels are all negatives/positives);
- fix degenerate case in `precision.at.given.recall.levels.over.classes` (labels in a fold are all negatives/positives);
- fix degenerate case in `do.stratified.cv.data.single.class` (sampling of the labels with just one positive/negative);
- add input variable `compute.performance` to the following high level functions:
  - `Do.TPR.DAG` and `Do.TPR.DAG.holdout`;
  - `Do.HTD` and `Do.HTD.holdout`;
  - `Do.GPAV` and `Do.GPAV.holdout`;
  - `Do.heuristic.methods` and `Do.heuristic.methods.holdout`;

10.7.2 Changes

- improve manual;

10.8 HEMDAG 2.3.6

10.8.1 New Features

- add `lexicographical.topological.sort`;

10.8.2 Changes

- fix minor bugs;
- improve manual;

10.9 HEMDAG 2.2.5

10.9.1 New Features

- precision-recall performance computed through `precrec` package:
  - add `precision.at.all.recall.levels.single.class`;
  - `PXR.at.multiple.recall.levels.over.classes` –> `precision.at.given.recall.levels.over.classes`;
- improve IO functions: the extension of the input or output file can be or plain text (`.txt`) or compressed (`.gz`);

10.9.2 Changes

- fix minor bugs;
- improve manual;

10.10 HEMDAG 2.2.4

10.10.1 Changes

- fix `CRAN` Package Check Results: remove unneeded header and define from `GPAV C++` source code

10.11 HEMDAG 2.2.3

10.11.1 New Features

- add `GPAV` algorithm (Burdakov et al., *Journal of Computational Mathematics*, 2006 – link);
- Embed `GPAV` algorithm in the top-down step of the functions `TPR.DAG`, `Do.TPR.DAG` and `Do.TPR.DAG.holdout`;
- Some functions have been defunct. To know the defunct functions just typing in the R environment: `help("HEMDAG-defunct");`

10.11.2 Changes

- improve manual;

10.11.3 AUTHOR

- add **Alessandro Petrini** as author for his contribution in writing the `C++` code of `GPAV` algorithm;

10.12 HEMDAG 2.1.3

10.12.1 Changes

- various fixes from 2.1.2

10.13 HEMDAG 2.1.2

10.13.1 New Features

- improve performance metrics:
  - add `compute.Fmeasure.multilabel`;
  - add `PXR.at.multiple.recall.levels.over.classes`;
  - all the performance metrics (`AUPRC`, `AUROC`, `FMM`, `PXR`) can be computed either **one-shot** or averaged **across folds**;
- improve the high-level hierarchical ensemble functions:
  - embed the new performance metric functions;
  - add the parameter `metric`: maximization by `FMAX` or `PRC` (see manual for further details);
  - add some checkers (warning/stop messages) to make the library more user-friendly;

10.13.2 Changes

- improve manual;

10.14 HEMDAG 2.0.1

10.14.1 Changes

- fix bug in `do.stratified.cv.data.single.class`;

10.15 HEMDAG 2.0.0

10.15.1 New Features

- add `TPR-DAG`: function gathering several hierarchical ensemble variants;
- add `Do.TPR.DAG`: high-level function to run `TPR-DAG` **cross-validated** experiments;
- add `Do.TPR.DAG.holdout`: high-level functions to run `TPR-DAG` **holdout** experiments;
- The following `TPR-DAG` and `DESCENS` high-level functions were remove:
  - `Do.tpr.threshold.free`;
  - `Do.tpr.threshold.cv`;

- `Do.tpr.weighted.threshold.free.cv`;
- `Do.tpr.weighted.threshold.cv`;
- `Do.descens.threshold.free`;
- `Do.descens.threshold.cv`;
- `Do.descens.weighted.threshold.free.cv`;
- `Do.descens.tau.cv`;
- `Do.descens.weighted.threshold.cv`;
- `Do.tpr.threshold.free.holdout`;
- `Do.tpr.threshold.holdout`;
- `Do.tpr.weighted.threshold.free.holdout`;
- `Do.tpr.weighted.threshold.holdout`;
- `Do.descens.threshold.free.holdout`;
- `Do.descens.threshold.holdout`;
- `Do.descens.weighted.threshold.free.holdout`;
- `Do.descens.tau.holdout`;
- `Do.descens.weighted.threshold.holdout`;

NOTE: all the removed functions can be run opportunely by setting the input parameters of the new high-level function `Do.TPR.DAG` (for **cross-validated** experiments) and `Do.TPR.DAG.holdout` (for **hold-out** experiments);

10.15.2 Changes

- improve manual;

10.16 HEMDAG 1.1.1

10.16.1 New Features

- add `DESCENS` algorithm;
- add Heuristic Methods `Max`, `And`, `Or` (Obozinski et al., Genome Biology, 2008 – link);
- add `tupla.matrix` function;

10.16.2 Changes

- improve manual;
- add link to the GitHub repository `HPOparser` (note: from version `2.6.0` `HPOparser` was changed in `obogaf::parser`);
- add `CITATION` file;

10.17 HEMDAG 1.0.0

10.17.1 Package Genesis

CHAPTER 11

HEMDAG License

11.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

11.2 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

11.3 TERMS AND CONDITIONS

11.3.1 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

11.3.2 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

11.3.3 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

11.3.4 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

11.3.5 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

11.3.6 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

11.3.7 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

11.3.8 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

11.3.9 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

11.3.10 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

11.3.11 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11.3.12 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

11.3.13 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

11.3.14 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

11.3.15 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

11.3.16 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

11.3.17 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRO-

GRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

11.3.18 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

11.4 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

Bibliography

[1] Oleg Burdakov, Oleg Sysoev, Anders Grimvall, and Mohamed Hussian. *An O(n2) Algorithm for Isotonic Regression*, chapter 83, pages 25–33. Springer US, Boston, MA, 2006. URL: https://doi.org/10.1007/0-387-30065-1_3, doi:10.1007/0-387-30065-1_3.

[2] Yuxiang Jiang, Tal Ronnen Oron, Wyatt T. Clark, Asma R. Bankapur, Daniel D'Andrea, Rosalba Lepore, Christopher S. Funk, Indika Kahanda, Karin M. Verspoor, Asa Ben-Hur, Da Chen Emily Koo, Duncan Penfold-Brown, Dennis Shasha, Noah Youngs, Richard Bonneau, Alexandra Lin, Sayed M. E. Sahraeian, Pier Luigi Martelli, Giuseppe Profiti, Rita Casadio, Renzhi Cao, Zhaolong Zhong, Jianlin Cheng, Adrian Altenhoff, Nives Skunca, Christophe Dessimoz, Tunca Dogan, Kai Hakala, Suwisa Kaewphan, Farrokh Mehryary, Tapio Salakoski, Filip Ginter, Hai Fang, Ben Smithers, Matt Oates, Julian Gough, Petri Törönen, Patrik Koskinen, Liisa Holm, Ching-Tai Chen, Wen-Lian Hsu, Kevin Bryson, Domenico Cozzetto, Federico Minneci, David T. Jones, Samuel Chapman, Dukka BKC, Ishita K. Khan, Daisuke Kihara, Dan Ofer, Nadav Rappoport, Amos Stern, Elena Cibrian-Uhalte, Paul Denny, Rebecca E. Foulger, Reija Hieta, Duncan Legge, Ruth C. Lovering, Michele Magrane, Anna N. Melidoni, Prudence Mutowo-Meullenet, Klemens Pichler, Aleksandra Shypitsyna, Biao Li, Pooya Zakeri, Sarah ElShal, Léon-Charles Tranchevent, Sayoni Das, Natalie L. Dawson, David Lee, Jonathan G. Lees, Ian Sillitoe, Prajwal Bhat, Tamás Nepusz, Alfonso E. Romero, Rajkumar Sasidharan, Haixuan Yang, Alberto Paccanaro, Jesse Gillis, Adriana E. Sedeño-Cortés, Paul Pavlidis, Shou Feng, Juan M. Cejuela, Tatyana Goldberg, Tobias Hamp, Lothar Richter, Asaf Salamov, Toni Gabaldon, Marina Marcet-Houben, Fran Supek, Qingtian Gong, Wei Ning, Yuanpeng Zhou, Weidong Tian, Marco Falda, Paolo Fontana, Enrico Lavezzo, Stefano Toppo, Carlo Ferrari, Manuel Giollo, Damiano Piovesan, Silvio C.E. Tosatto, Angela del Pozo, José M. Fernández, Paolo Maietta, Alfonso Valencia, Michael L. Tress, Alfredo Benso, Stefano Di Carlo, Gianfranco Politano, Alessandro Savino, Hafeez Ur Rehman, Matteo Re, Marco Mesiti, Giorgio Valentini, Joachim W. Bargsten, Aalt D. J. van Dijk, Branislava Gemovic, Sanja Glisic, Vladmir Perovic, Veljko Veljkovic, Nevena Veljkovic, Danillo C. Almeida-e-Silva, Ricardo Z. N. Vencio, Malvika Sharan, Jörg Vogel, Lakesh Kansakar, Shanshan Zhang, Slobodan Vucetic, Zheng Wang, Michael J. E. Sternberg, Mark N. Wass, Rachael P. Huntley, Maria J. Martin, Claire O'Donovan, Peter N. Robinson, Yves Moreau, Anna Tramontano, Patricia C. Babbitt, Steven E. Brenner, Michal Linial, Christine A. Orengo, Burkhard Rost, Casey S. Greene, Sean D. Mooney, Iddo Friedberg, and Predrag Radivojac. An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome Biology*, 17(1):184, Sep 2016. URL: https://doi.org/10.1186/s13059-016-1037-6, doi:10.1186/s13059-016-1037-6.

[3] Marco Notaro, Max Schubach, Peter N. Robinson, and Giorgio Valentini. Prediction of Human Phenotype Ontology terms by means of hierarchical ensemble methods. *BMC Bioinformatics*, 18(1):449, December 2017. URL: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-017-1854-y (visited on 2021-02-01), doi:10.1186/s12859-017-1854-y.

[4] G. Obozinski, G. Lanckriet, C. Grant, Jordan. M., and W.S. Noble. Consistent probabilistic output for protein function prediction. *Genome Biology*, 9:135–142, 2008. URL: http://europepmc.org/articles/PMC2447540, doi:10.1186/gb-2008-9-s1-s6.

[5] Giorgio Valentini. True path rule hierarchical ensembles for genome-wide gene function prediction. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(3):832–847, 2011. URL: https://doi.org/10.1109/TCBB.2010.38, doi:10.1109/TCBB.2010.38.