obogaf::parser Documentation

Marco Notaro

Mar 29, 2021

Installation Getting Started

`obogaf::parser` is a perl5 module designed to handle **GO** and **HPO** obo file and their gene annotation file (gaf file). However, all the `obogaf::parser` subroutines can be safely used to parse any *obo* file listed in OBO foundry and any gene *annotation* file structured as those shown in GOA website and HPO website – basically a `csv` file using `tab` as separator).

Subroutines contained in `obogaf::parser`:

- **build_edges**: extract edges from an `obo` file;
- **build_subonto**: extract edges for a specific subontology domain;
- **make_stat**: make basic statistic on a graph;
- **get_parents_or_children_list**: build parents or children list for each node of the graph;
- **obo_filter**: prune obo file relatively to a set of given ontology terms;
- **gene2biofun**: build the annotations list from a gaf file;
- **map_OBOterm_between_release**: map ontology terms between releases;

To call an `obogaf::parser` subroutine you must preface the subroutine's name with the name of the library (`obogaf::parser`) and double colon (`::`): `obogaf::parser::subroutine-to-call`. See examples in *Tutorial* section for more details.

CHAPTER 1

Quickstart

This short *HowTo* guides you from downloading `obogaf::parser` to make your first parsing with `obogaf::parser`.

1.1 Installation

Please goto the *Installation* section and chose one of the shown ways to install `obogaf::parser` (we suggest to use the *Installation via Conda* option).

1.2 Load obogaf::parser library

To load `obogaf::parser` module, just type inside a Perl script `use obogaf::parser`. More precisely, the *header* of your Perl script should be:

```perl
#!/usr/bin/perl

use strict;
use warnings;

use obogaf::parser;

... beginning of your perl code ...
```

To run the Perl script you can make it executable by typing `chmod +x perl-script.pl` or by prefacing the script with the Perl interpreter (`perl perl-script.pl`).

1.3 Your first parsing

Let us use `obogaf::parser` to extract edges (in the form `source-destination`) from the Gene Ontology (GO) obo file. Firstly we must download the GO obo file from the Gene Ontoloy website. Then we use `obogaf::parser` to extract edges.

```
## perl shebang (unix)
#!/usr/bin/perl

## load the module
use obogaf::parser;

## download GO obo file
my $obofile= "gobasic.obo";
my $gobo= qx{wget --output-document=$obofile http://purl.obolibrary.org/obo/go/go-
↪basic.obo};
print "GO obo file downloaded: done\n\n";

## extract and print GO edges
my $gores= obogaf::parser::build_edges($obofile);
print "${$gores}";
```

`obogaf::parser` can do much more than that! Go to the *Tutorial* section to discover what this module can do! But first get a look to *Installation* section . . .

CHAPTER 2

Installation

obogaf::parser is available on CPAN as well as through Bioconda and also from source code. You can use one of the following ways for installing obogaf::parser.

2.1 Installation via Conda

This is the recommended way to install obogaf::parser for normal user because it will enable you to switch software versions easily and in addition Perl 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 *Python 3* version of Miniconda.
- Answer yes to the question whether conda shall be put into your PATH.

Then, you can install obogaf::parser with

```
conda install -c bioconda perl-obogaf-parser
```

from the Bioconda channel.

2.2 Global Installation

You can directly install the module via cpan:

```
$ cpan install obogaf::parser
```

or via cpanm:

```
$ cpanm obogaf::parser
```

make sure to install `cpanm` before running the command above

Note: to install the `obogaf::parser` globally you must be a root-user

2.3 Local Installation

If you do not have root permit, you can clone (or download) the `obogaf::parser` git repository (link) and initialize your Perl script as follow:

```
#!/usr/bin/perl

use strict;
use warnings;
use lib 'path/to/folder/containing/obogaf-parser-module'; ## nb: folder != full␣
↪filename
use parser;

... beginning of your Perl code ...
```

2.4 Installing from Source

To build `obogaf::parser` from scratch follow the command shown below:

```
$ cd ~;
$ git clone git://github.com/marconotaro/obogaf-parser.git obogaf-parser;
$ cd obogaf-parser;

$ perl Makefile.PL;
$ make manifest;
$ make;
$ make test;
$ sudo make install;
$ make veryclean; ## to clean built files
```

2.5 Dependencies

For building `obogaf::parser` you will need the following dependencies

- Perl (>= v5.22.1)
- Perl Module:
  - Graph - graph data structures and algorithms
  - PerlIO::gzip - Perl extension to provide a PerlIO layer to gzip/gunzip
- Test Module:
  - Test::More - yet another framework for writing test scripts
  - Test::Exception - Test exception-based code

    - Test::Files - A Test::Builder based module to ease testing with files and dirs

- Configure Module:
    - Module::Metadata - Gather package and POD information from perl module files
    - ExtUtils::MakeMaker - Create a module Makefile

CHAPTER 3

Usage of obogaf::parser

For a detailed description of available subroutines contained in `obogaf::parser` perl module please go to the CPAN page https://metacpan.org/pod/obogaf::parser and have a look to the *reference manual*. Alternatively, after installing the module (see *Installation* section), you can type on terminal `perldoc obogaf::parser` to get a glimpse of the *reference manual*.

CHAPTER 4

Tutorial

Here we show a step-by-step application of `obogaf::parser` by using the Gene Ontology (GO) and the Human Phenotype Ontology (HPO) and their respective annotation file. The snippets of Perl code shown in the examples below are glued together respectively in the script `GOscript.pl` and `HPOscript.pl` shown in the page *Scripts*.

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

- obogaf::parser >= 1.373
- Perl >= 5.22.1
- Ubuntu >= 16.04

4.1 Gene Ontology (GO)

For all the examples shown in this tutorial, we store I/O files in the directory `data`:

```
$ cd ~ && mkdir -p data/  ## create a directory if it does not already exist
```

4.1.1 Parse the GO obo file

First of all we must download the *obo* file from the Gene Ontoloy website. We download the *basic* version of the GO, because this version excludes relationships that cross the 3 GO hierarchies (BP, MF, CC). To do that in a Linux environment, just type on the bash:

```
$ cd data/ && wget http://purl.obolibrary.org/obo/go/go-basic.obo -O gobasic.obo
```

Let's have a look to the `gobasic.obo` (release `2019-10-07`) file to see how it is structured. For instance, to display the first `60` lines we can type on the Linux Shell `head -n60 data/gobasic.obo`:

```
format-version: 1.2
data-version: releases/2019-10-07
subsetdef: gocheck_do_not_annotate "Term not to be used for direct annotation"
subsetdef: gocheck_do_not_manually_annotate "Term not to be used for direct manual␣
↪annotation"
subsetdef: goslim_agr "AGR slim"
subsetdef: goslim_aspergillus "Aspergillus GO slim"
subsetdef: goslim_candida "Candida GO slim"
subsetdef: goslim_chembl "ChEMBL protein targets summary"
subsetdef: goslim_flybase_ribbon "FlyBase Drosophila GO ribbon slim"
subsetdef: goslim_generic "Generic GO slim"
subsetdef: goslim_metagenomics "Metagenomics GO slim"
subsetdef: goslim_mouse "Mouse GO slim"
subsetdef: goslim_pir "PIR GO slim"
subsetdef: goslim_plant "Plant GO slim"
subsetdef: goslim_pombe "Fission yeast GO slim"
subsetdef: goslim_synapse "synapse GO slim"
subsetdef: goslim_yeast "Yeast GO slim"
synonymtypedef: syngo_official_label "label approved by the SynGO project"
synonymtypedef: systematic_synonym "Systematic synonym" EXACT
default-namespace: gene_ontology
remark: cvs version: use data-version
remark: Includes Ontology(OntologyID(OntologyIRI(<http://purl.obolibrary.org/obo/go/
↪never_in_taxon.owl>))) [Axioms: 18 Logical Axioms: 0]
ontology: go

[Term]
id: GO:0000001
name: mitochondrion inheritance
namespace: biological_process
def: "The distribution of mitochondria, including the mitochondrial genome, into␣
↪daughter cells after mitosis or meiosis, mediated by interactions between␣
↪mitochondria and the cytoskeleton." [GOC:mcc, PMID:10873824, PMID:11389764]
synonym: "mitochondrial inheritance" EXACT []
is_a: GO:0048308 ! organelle inheritance
is_a: GO:0048311 ! mitochondrion distribution

[Term]
id: GO:0000002
name: mitochondrial genome maintenance
namespace: biological_process
def: "The maintenance of the structure and integrity of the mitochondrial genome;␣
↪includes replication and segregation of the mitochondrial chromosome." [GOC:ai,␣
↪GOC:vw]
is_a: GO:0007005 ! mitochondrion organization

[Term]
id: GO:0000003
name: reproduction
namespace: biological_process
alt_id: GO:0019952
alt_id: GO:0050876
def: "The production of new individuals that contain some portion of genetic material␣
↪inherited from one or more parent organisms." [GOC:go_curators, GOC:isa_complete,␣
↪GOC:jl, ISBN:0198506732]
subset: goslim_agr
subset: goslim_chembl
```

(continues on next page)

(continued from previous page)

```
subset: goslim_flybase_ribbon
subset: goslim_generic
subset: goslim_pir
subset: goslim_plant
synonym: "reproductive physiological process" EXACT []
xref: Wikipedia:Reproduction
is_a: GO:0008150 ! biological_process

[Term]
id: GO:0000005
name: obsolete ribosomal chaperone activity

... to be continued ...
```

Let's imagine we would like to shrink the `gobasic.obo` file to a subset of terms we are interested in. What we have to do is storing the `obo` terms that we want to isolate in a plain text file and calling the `obo_filter` subroutine:

```
## store in a plain file the list of GO terms
my @terms = qw(GO:0000002 GO:0000003 GO:0000018 GO:0000030 GO:0000038);
my $termsfile= "data/goterms.txt";
open OUT, "> $termsfile";
foreach my $go (@terms){print OUT "$go\n";}
close OUT;

## shrink GO obo file to our list of terms
$res= obo_filter($obofile, $termsfile);
my $newobo= "data/go-shrunk.obo";
open OUT, ">", $newobo;
print OUT "${$res}";
close OUT;
```

The returned narrowed `obo` file looks as the following:

```
format-version: 1.2
data-version: releases/2019-10-07
subsetdef: gocheck_do_not_annotate "Term not to be used for direct annotation"
subsetdef: gocheck_do_not_manually_annotate "Term not to be used for direct manual␣
↪annotation"
subsetdef: goslim_agr "AGR slim"
subsetdef: goslim_aspergillus "Aspergillus GO slim"
subsetdef: goslim_candida "Candida GO slim"
subsetdef: goslim_chembl "ChEMBL protein targets summary"
subsetdef: goslim_flybase_ribbon "FlyBase Drosophila GO ribbon slim"
subsetdef: goslim_generic "Generic GO slim"
subsetdef: goslim_metagenomics "Metagenomics GO slim"
subsetdef: goslim_mouse "Mouse GO slim"
subsetdef: goslim_pir "PIR GO slim"
subsetdef: goslim_plant "Plant GO slim"
subsetdef: goslim_pombe "Fission yeast GO slim"
subsetdef: goslim_synapse "synapse GO slim"
subsetdef: goslim_yeast "Yeast GO slim"
synonymtypedef: syngo_official_label "label approved by the SynGO project"
synonymtypedef: systematic_synonym "Systematic synonym" EXACT
default-namespace: gene_ontology
remark: cvs version: use data-version
remark: Includes Ontology(OntologyID(OntologyIRI(<http://purl.obolibrary.org/obo/go/
↪never_in_taxon.owl>))) [Axioms: 18 Logical Axioms: 0]
```

(continues on next page)

(continued from previous page)

```
ontology: go

[Term]
id: GO:0000002
name: mitochondrial genome maintenance
namespace: biological_process
def: "The maintenance of the structure and integrity of the mitochondrial genome;␣
↪includes replication and segregation of the mitochondrial chromosome." [GOC:ai,␣
↪GOC:vw]
is_a: GO:0007005 ! mitochondrion organization

[Term]
id: GO:0000003
name: reproduction
namespace: biological_process
alt_id: GO:0019952
alt_id: GO:0050876
def: "The production of new individuals that contain some portion of genetic material␣
↪inherited from one or more parent organisms." [GOC:go_curators, GOC:isa_complete,␣
↪GOC:jl, ISBN:0198506732]
subset: goslim_agr
subset: goslim_chembl
subset: goslim_flybase_ribbon
subset: goslim_generic
subset: goslim_pir
subset: goslim_plant
synonym: "reproductive physiological process" EXACT []
xref: Wikipedia:Reproduction
is_a: GO:0008150 ! biological_process

[Term]
id: GO:0000018
name: regulation of DNA recombination
namespace: biological_process
def: "Any process that modulates the frequency, rate or extent of DNA recombination,␣
↪a DNA metabolic process in which a new genotype is formed by reassortment of genes␣
↪resulting in gene combinations different from those that were present in the␣
↪parents." [GOC:go_curators, ISBN:0198506732]
is_a: GO:0051052 ! regulation of DNA metabolic process
relationship: regulates GO:0006310 ! DNA recombination

[Term]
id: GO:0000030
name: mannosyltransferase activity
namespace: molecular_function
def: "Catalysis of the transfer of a mannosyl group to an acceptor molecule,␣
↪typically another carbohydrate or a lipid." [GOC:ai, GOC:cjm]
xref: Reactome:R-HSA-162797 "mannose (a1-2) mannose (a1-6) (ethanolamineP) mannose␣
↪(a1-4) glucosaminyl-acyl-PI -> mannose (a1) mannose (a1-2) mannose (a1-6)␣
↪(ethanolamineP) mannose (a1-4) glucosaminyl-acyl-PI"
xref: Reactome:R-HSA-162830 "glucosaminyl-acyl-PI + dolichol phosphate D-mannose ->␣
↪mannose(a1l-4)glucosaminyl-acyl-PI + dolichol phosphate"
xref: Reactome:R-HSA-446198 "ALG12 transfers Man to N-glycan precursor (GlcNAc)2␣
↪(Man)7 (PP-Dol)1"
xref: Reactome:R-HSA-4720497 "Defective ALG12 does not add mannose to the N-glycan␣
↪precursor"
is_a: GO:0016758 ! transferase activity, transferring hexosyl groups
```

(continues on next page)

(continued from previous page)

```
[Term]
id: GO:0000038
name: very long-chain fatty acid metabolic process
namespace: biological_process
def: "The chemical reactions and pathways involving a fatty acid which has a chain␣
↪length greater than C22." [CHEBI:27283, GOC:hjd]
synonym: "very long chain fatty acid metabolic process" EXACT [GOC:bf]
synonym: "very-long-chain fatty acid metabolic process" EXACT []
synonym: "very-long-chain fatty acid metabolism" EXACT []
is_a: GO:0006631 ! fatty acid metabolic process
```

To extrapolate the GO edges from the `gobasic.obo` file, we can use the subroutine `build_edges`. This subroutine receives in input the `obo` file:

```
## loading the obo file and calling the subroutine
my $obofile= "data/gobasic.obo";
my $gores= obogaf::parser::build_edges($obofile);

## storing
my $goedges= "data/gobasic-edges.txt";
open OUT, "> $goedges";
print OUT "${$gores}"; ## dereferencing
close OUT;
```

For the sake of the space, below we just show the first `25` lines of the output file `gobasic-edges.txt` (`head -n25 data/gobasic-edges.txt`):

```
biological_process   GO:0048308   GO:0000001   organelle inheritance   mitochondrion␣
↪inheritance  is-a
biological_process   GO:0048311   GO:0000001   mitochondrion distribution mitochondrion␣
↪inheritance  is-a
biological_process   GO:0007005   GO:0000002   mitochondrion organization mitochondrial␣
↪genome maintenance is-a
biological_process   GO:0008150   GO:0000003   biological_process   reproduction   is-a
molecular_function   GO:0005385   GO:0000006   zinc ion transmembrane transporter␣
↪activity  high-affinity zinc transmembrane transporter activity is-a
molecular_function   GO:0005385   GO:0000007   zinc ion transmembrane transporter␣
↪activity  low-affinity zinc ion transmembrane transporter activity is-a
molecular_function   GO:0000030   GO:0000009   mannosyltransferase activity  alpha-1,6-
↪mannosyltransferase activity is-a
molecular_function   GO:0016765   GO:0000010   transferase activity, transferring alkyl␣
↪or aryl (other than methyl) groups   trans-hexaprenyltranstransferase activity is-a
biological_process   GO:0007033   GO:0000011   vacuole organization vacuole inheritance␣
↪ is-a
biological_process   GO:0048308   GO:0000011   organelle inheritance   vacuole␣
↪inheritance  is-a
biological_process   GO:0006281   GO:0000012   DNA repair  single strand break repair␣
↪is-a
molecular_function   GO:0004520   GO:0000014   endodeoxyribonuclease activity   single-
↪stranded DNA endodeoxyribonuclease activity is-a
cellular_component   GO:1902494   GO:0000015   catalytic complex phosphopyruvate␣
↪hydratase complex   is-a
cellular_component   GO:0005829   GO:0000015   cytosol  phosphopyruvate hydratase␣
↪complex   part-of
molecular_function   GO:0004553   GO:0000016   hydrolase activity, hydrolyzing O-
↪glycosyl compounds  lactase activity  is-a
```

(continues on next page)

(continued from previous page)

```
biological_process   GO:0042946  GO:0000017  glucoside transport  alpha-glucoside␣
↪transport  is-a
biological_process   GO:0051052  GO:0000018  regulation of DNA metabolic process␣
↪regulation of DNA recombination  is-a
biological_process   GO:0000018  GO:0000019  regulation of DNA recombination ␣
↪regulation of mitotic recombination is-a
biological_process   GO:0051231  GO:0000022  spindle elongation   mitotic spindle␣
↪elongation is-a
biological_process   GO:1903047  GO:0000022  mitotic cell cycle process mitotic␣
↪spindle elongation is-a
biological_process   GO:0000070  GO:0000022  mitotic sister chromatid segregation  ␣
↪mitotic spindle elongation part-of
biological_process   GO:0007052  GO:0000022  mitotic spindle organization  mitotic␣
↪spindle elongation part-of
biological_process   GO:0005984  GO:0000023  disaccharide metabolic process   maltose␣
↪metabolic process  is-a
biological_process   GO:0000023  GO:0000024  maltose metabolic process  maltose␣
↪biosynthetic process  is-a
biological_process   GO:0046351  GO:0000024  disaccharide biosynthetic process  ␣
↪maltose biosynthetic process  is-a

... to be continued ...
```

The first column of the output file refers to the domain whose a GO term belong to, the second and the third column represent the edge as pair of nodes in the form `source (parent) - destination (child)`, the fourth and the fifth column are the name of the source and destination obo term ID and the sixth column refers to the kind of relationships. This column can assume only two values, `is-a` and `part-of`, since it is safe grouping annotations by using both these relationships. For more details about GO relationships have a look at this link.

To isolate nodes and relationships belonging to one of the GO sub-ontology (e.g. `biological_process (BP)`), we can use the subroutine `build_subonto`. This subroutine receives in input the edges file obtained by calling `build_edges` and the specific sub-domain for which we want to extrapolate edges.

```
## loading and calling
my $goedges= "data/gobasic-edges.txt"; ## obtained previously by calling␣
↪obogaf::parser::build_edges
my $BPres= obogaf::parser::build_subonto($goedges, "biological_process");

## storing
my $BPedges= "data/gobasic-edgesBP.txt";
open OUT, "> $BPedges";
print OUT "${$BPres}";
close OUT;
```

Below we report the first 10 lines of `gobasic-edgesBP.txt` (`head -n10 data/gobasic-edgesBP.txt`):

```
GO:0048308  GO:0000001  organelle inheritance   mitochondrion inheritance  is-a
GO:0048311  GO:0000001  mitochondrion distribution mitochondrion inheritance  is-a
GO:0007005  GO:0000002  mitochondrion organization mitochondrial genome maintenance␣
↪is-a
GO:0008150  GO:0000003  biological_process   reproduction   is-a
GO:0007033  GO:0000011  vacuole organization vacuole inheritance  is-a
GO:0048308  GO:0000011  organelle inheritance   vacuole inheritance  is-a
GO:0006281  GO:0000012  DNA repair  single strand break repair is-a
GO:0042946  GO:0000017  glucoside transport  alpha-glucoside transport  is-a
```

(continues on next page)

(continued from previous page)

```
GO:0051052  GO:0000018  regulation of DNA metabolic process regulation of DNA␣
↪recombination  is-a
GO:0000018  GO:0000019  regulation of DNA recombination  regulation of mitotic␣
↪recombination is-a

... to be continued ...
```

It is worth noting that the same output can be also achieved by using the `grep` command (in a Linux environment):

```
$ grep "biological_process" data/gobasic-edges.txt | cut -f2- > data/gobasic-edgesBP.
↪txt
```

If we want to isolate nodes and relationships separately for each GO subontology at one fell swoop, by Perl:

```
my $goedges= "data/gobasic-edges.txt"; ## obtained previously by calling␣
↪obogaf::parser::build_edges
my @domains= qw(biological_process molecular_function cellular_component);
my %aspects=(biological_process => "BP", molecular_function => "MF", cellular_
↪component => "CC");

foreach my $domain (@domains){
    my $outfile= "data/gobasic-edges"."$aspects{$domain}".".txt";
    open OUT, "> $outfile";
    my $domainres= obogaf::parser::build_subonto($goedges, $domain);
    print OUT "${$domainres}";
    close OUT;
}
```

and by bash:

```
goedges="data/gobasic-edges.txt"; ## obtained previously by calling␣
↪obogaf::parser::build_edges
domains=("biological_process" "molecular_function" "cellular_component");
aspects=("BP" "MF" "CC");

len="${#domains[@]}";
for ((i = 0 ; i < len ; i++)); do
    grep ${domains[$i]} data/gobasic-edges.txt | cut -f2- > data/gobasic-edges$
↪{aspects[$i]}.txt
done
```

To print some statistics on the GO graph, we can use the subroutine `make_stat`. The input arguments required by this subroutine are:

1. `$goedges`: file containing the GO graph represented as a list of edges where each edge is turn represented as a pair of vertices `tab` separated (`$goedges` file can be obtained by calling the `build_edges` subroutine)
2. `$parentIndex` and `$childIndex`: index referring restrictively to the column containing the `source` and `destination` nodes in the `$goedges` file (reminder: Perl starts counting from zero).

```
my ($goedges, $parentIndex, $childIndex)= ("data/gobasic-edges.txt", 1, 2);
my $res= obogaf::parser::make_stat($goedges, $parentIndex, $childIndex);
print "$res";

## results printed on the shell
#oboterm <tab> degree <tab> indegree <tab> outdegree
GO:0032991  469   1  468
```

(continues on next page)

(continued from previous page)

```
GO:0110165  436   1  435
GO:0016616  346   1  345
GO:0016709  303   2  301
GO:0016758  204   1  203
GO:0048856  199   1  198
GO:0098797  181   2  179
GO:0003006  172   2  170
GO:0005737  171   2  169
GO:0016747  159   1  158
.
.
.
~summary stat~
nodes: 44733
edges: 82705
max degree: 469
min degree: 1
median degree: 2.0000
average degree: 1.8489
density: 4.1332e-05
```

As we can observe from the snippet above, for each node of the graph, `degree`, `in-degree` and `out-degree` are printed. Nodes are sorted in a decreasing order on the basis of degree, from the higher to the smaller one. In addition the following statistics are also returned: 1) number of nodes and edges of the graph; 2) maximum and minimum degree; 3) average and median degree; 4) density of the graph.

To compute the stats just for a specific GO subontology (e.g. `GO BP`) we can always use `make_stat`, by properly setting its input arguments:

```
my ($goedges, $parentIndex, $childIndex)= ("data/gobasic-edgesBP.txt", 0, 1);
my $res= obogaf::parser::make_stat($goedges, $parentIndex, $childIndex);
print "$res";

## results returned on the shell
oboterm <tab> degree <tab> indegree <tab> outdegree
#oboterm <tab> degree <tab> indegree <tab> outdegree
GO:0048856  199   1  198
GO:0003006  172   2  170
GO:0051241  136   2  134
GO:0051240  129   2  127
GO:0014070  128   1  127
GO:1901700  112   1  111
GO:0022414  110   2  108
GO:0048646  108   2  106
GO:0031328  105   3  102
GO:1901361  105   2  103
.
.
.
~summary stat~
nodes: 29457
edges: 62232
max degree: 199
min degree: 1
median degree: 3.0000
average degree: 2.1126
density: 7.1722e-05
```

obogaf::parser computes also the parents and children list for each node of the graph:

```
my $parlist= "gobasic-parGO.txt";
my ($goedges, $parentIndex, $childIndex)= ("data/gobasic-edges.txt", 1, 2);
my $pares= obogaf::parser::get_parents_or_children_list($goedges, $parentIndex,
↪$childIndex, "parents");
open FH, "> $parlist";
foreach my $k (sort{$a cmp $b} keys %$pares) { print FH "$k $$pares{$k}\n";} ##␣
↪parents  list
close FH;

my $chdlist= "gobasic-chdGO.txt";
my $chdres= obogaf::parser::get_parents_or_children_list($goedges, $parentIndex,
↪$childIndex, "children");
open FH, "> $chdlist";
foreach my $k (sort{$a cmp $b} keys %$chdres) { print FH "$k $$chdres{$k}\n";} ##␣
↪children list
close FH;
```

Below we show few lines of gobasic-parGO.txt as example:

```
GO:0000001 GO:0048308|GO:0048311
GO:0000002 GO:0007005
GO:0000003 GO:0008150
GO:0000006 GO:0005385
GO:0000007 GO:0005385
GO:0000009 GO:0000030
GO:0000010 GO:0016765
GO:0000011 GO:0007033|GO:0048308
GO:0000012 GO:0006281
GO:0000014 GO:0004520
GO:0000015 GO:0005829|GO:1902494
GO:0000016 GO:0004553
GO:0000017 GO:0042946
GO:0000018 GO:0051052
GO:0000019 GO:0000018
GO:0000022 GO:0000070|GO:0007052|GO:0051231|GO:1903047
GO:0000023 GO:0005984
GO:0000024 GO:0000023|GO:0046351
GO:0000025 GO:0000023|GO:0046352
GO:0000026 GO:0000030

... to be continued ...
```

The first column contains a GO term whereas the second one contains the list (pipe separated) of its parent terms. The file gobasic-chdGO.txt has the same structure, but instead of parents list contains the children list.

Obviously, obogaf::parser::get_parents_or_children_list can also be run on a subontology file (e.g. gobasic-edgesBP.txt). The only thing to do is to proper set the parameters $parentIndex and $childIndex.

4.1.2 Parse the GOA annotation file

obogaf::parser can be also used to parse the annotation file taken from the Gene Ontology Annotation (GOA) Database (link).

For the examples shown below we use the annotation file of the CHICKEN model organism (release 7/29/19), but of course obogaf::parser subroutines can be applied to parse the annotation file of any other organisms listed in the GOA database and more in general to parse any file structured as those listed in the GOA database.

NOTE: the annotation file on GOA website are monthly updated. The release used at the time of writing this tutorial is July release (2019-11-11).

First we must download the annotation file in the data folder (note that the link show below refers to the most updated release):

```
$ cd data && wget ftp://ftp.ebi.ac.uk/pub/databases/GO/goa/CHICKEN/goa_chicken.gaf.gz␣
↪-O goa_chicken.gaf.gz
```

By having a look to the goa_chicken.gaf.gz file we see that it is structured as follow (for the sake of space we display just the first 20 lines):

```
!gaf-version: 2.1
!
!The set of protein accessions included in this file is based on UniProt reference␣
↪proteomes, which provide one protein per gene.
!They include the protein sequences annotated in Swiss-Prot or the longest TrEMBL␣
↪transcript if there is no Swiss-Prot record.
!If a particular protein accession is not annotated with GO, then it will not appear␣
↪in this file.
!
!Note that the annotation set in this file is filtered in order to reduce redundancy;␣
↪the full, unfiltered set can be found in
!ftp://ftp.ebi.ac.uk/pub/databases/GO/goa/UNIPROT/goa_uniprot_all.gz
!
!Generated: 2019-11-11 15:58
!GO-version: http://purl.obolibrary.org/obo/go/releases/2019-11-09/extensions/go-plus.
↪owl
!
UniProtKB   A0A088BIK7  EDbeta   GO:0005200  GO_REF:0000002 IEA   InterPro:IPR003461 ␣
↪ F  Keratin  EDbeta|EDBETA  protein  taxon:9031  20191109 InterPro
UniProtKB   A0A088BIK7  EDbeta   GO:0005882  GO_REF:0000038 IEA   UniProtKB-KW:KW-
↪0416 C  Keratin  EDbeta|EDBETA  protein  taxon:9031  20191109 UniProt
UniProtKB   A0A088BIK7  EDbeta   GO:0007010  GO_REF:0000108 IEA   GO:0005200  P ␣
↪Keratin  EDbeta|EDBETA  protein  taxon:9031  20191109 GOC
UniProtKB   A0A0A0MQ32  LOXL2    GO:0000122  GO_REF:0000107 IEA  ␣
↪UniProtKB:Q9Y4K0|ensembl:ENSP00000373783  P  Lysyl oxidase homolog 2 LOXL2 protein ␣
↪taxon:9031  20191109 Ensembl
UniProtKB   A0A0A0MQ32  LOXL2    GO:0000785  GO_REF:0000107 IEA  ␣
↪UniProtKB:Q9Y4K0|ensembl:ENSP00000373783  C  Lysyl oxidase homolog 2 LOXL2 protein ␣
↪taxon:9031  20191109 Ensembl
UniProtKB   A0A0A0MQ32  LOXL2    GO:0001666  GO_REF:0000107 IEA  ␣
↪UniProtKB:P58022|ensembl:ENSMUSP00000022660  P  Lysyl oxidase homolog 2 LOXL2␣
↪protein  taxon:9031  20191109 Ensembl
UniProtKB   A0A0A0MQ32  LOXL2    GO:0001837  GO_REF:0000107 IEA  ␣
↪UniProtKB:Q9Y4K0|ensembl:ENSP00000373783  P  Lysyl oxidase homolog 2 LOXL2 protein ␣
↪taxon:9031  20191109 Ensembl
UniProtKB   A0A0A0MQ32  LOXL2    GO:0001935  GO_REF:0000107 IEA  ␣
↪UniProtKB:Q9Y4K0|ensembl:ENSP00000373783  P  Lysyl oxidase homolog 2 LOXL2 protein ␣
↪taxon:9031  20191109 Ensembl

... to be continued ...
```

Now we can build the list of annotations by using the subroutine gene2biofun. The input arguments required are:

1. `$inputfile`: GOA annotation file for the `CHICKEN` organism;
2. `$geneindex`: and `$geneindex`: index referring respectively to the column containing the proteins and the GO term in the `$inputfile` file.

```
my ($inputfile, $geneindex, $classindex)= ("data/goa_chicken.gaf.gz", 1, 4);
my ($res, $stat)= obogaf::parser::gene2biofun($inputfile, $geneindex, $classindex);

my $goaout= "data/chicken.uniprot2go.txt";
open OUT, "> $goaout";
foreach my $k (sort{$a cmp $b} keys %$res) { print OUT "$k $$res{$k}\n";}
close OUT;
print "${$stat}\n";

## results printed on the shell
genes: 15695
ontology terms: 13953
```

`gene2biofun` returns a list of two anonymous references. The first is an anonymous hash storing for each UniProtKB protein all its associated GO terms (pipe separated). The second is an anonymous scalar containing basic statistics such as the total unique number of proteins and ontology terms. In the example above the anonymous hash is addressed in the output file `data/chicken.uniprot2go.txt` and the stats are printed on the shell. Finally, it is worth noting that `gene2biofun` can handle both compress `.gz` file and plain `.txt` file. Below we report as an example a snapshot of the associations between UniProtKB entry and GO terms obtained by running `gene2biofun` and stored in the file `data/chicken.uniprot2go.txt` (`head -n10 data/chicken.uniprot2go.txt`):

```
A0A088BIK7 GO:0005200|GO:0005882|GO:0007010
A0A0A0MQ32␣
↪GO:0000122|GO:0000785|GO:0001666|GO:0001837|GO:0001935|GO:0002040|GO:0004720|GO:0005044|GO:0005507
A0A0A0MQ34 GO:0009374
A0A0A0MQ35 GO:0000421|GO:0005654|GO:0005765|GO:0016021|GO:0032266|GO:0097352
A0A0A0MQ36 GO:0005246|GO:0005509|GO:0007165
A0A0A0MQ42 GO:0005654|GO:0005794|GO:0019221|GO:0030368
A0A0A0MQ45␣
↪GO:0000086|GO:0004674|GO:0005524|GO:0005634|GO:0005654|GO:0005813|GO:0007147|GO:0018105|GO:0032154
A0A0A0MQ47␣
↪GO:0000122|GO:0000993|GO:0002039|GO:0005634|GO:0005829|GO:0008285|GO:0010452|GO:0018024|GO:0018026
A0A0A0MQ52␣
↪GO:0000724|GO:0003678|GO:0003682|GO:0003688|GO:0003697|GO:0005524|GO:0005634|GO:0006270|GO:0007292
A0A0A0MQ56 GO:0005615|GO:0005623|GO:0005874|GO:0010975|GO:1990830

... to be continued...
```

4.1.3 Map GO terms between releases

In time-lapse hold-out experiments we use annotations of an old GO release to predict the protein function of a more recent GO release. However, between different GO releases some ontology terms could be removed, others changed or become obsolete. Then before beginning time-lapse hold-out experiments, we need to map the old GO terms to the new ones by parsing the annotation file of an *old* GO release using as **key** the *alt-ID* taken from the obo file of the *new* GO release . The subroutine `map_OBOterm_between_release` does that for us.

Firstly, we must download the old annotation file of the `CHICKEN` organism in the `data` directory (here we use the `07/06/16` release):

```
$ cd data && wget ftp://ftp.ebi.ac.uk/pub/databases/GO/goa/old/CHICKEN/goa_chicken.
↪gaf.128.gz -O goa_chicken.gaf.128.gz
```

The input arguments required by `map_OBOterm_between_release` are:

1. `$obofile`: the *new* release of a GO obo file (here we use the `01/07/19` release). This file is used to make the `alt_id - id` pairing by using `alt_id` as key;
2. `$goafileOld`: the *old* release of an annotation file (for this example we use `07/06/16` release);
3. `$classindex`: the index referring to the column of the `$goafileOld` containing the ontology terms to be mapped (in the `GOA` file the GO terms are in the 4 columns – NB: we must start to count from zero).

```
my ($obofile, $goafileOld, $classindex)= ("data/gobasic.obo", "data/goa_chicken.gaf.
↪128.gz", 4);
my ($res, $stat)= obogaf::parser::map_OBOterm_between_release($obofile, $goafileOld,
↪$classindex);

my $mapfile= "data/chicken.goa.mapped.txt";
open OUT, "> $mapfile";
print OUT "${$res}";
close OUT;
print "${$stat}";

# results printed on the shell
#alt-id <tab> id
GO:0000042  GO:0034067
GO:0000975  GO:0044212
GO:0000982  GO:0000981
GO:0000983  GO:0016251
GO:0001075  GO:0016251
GO:0001077  GO:0001228
GO:0001078  GO:0001227
GO:0001104  GO:0003712
GO:0001105  GO:0003713
GO:0001106  GO:0003714
.
.
.
Tot. ontology terms: 12546
Tot. altID: 2617
Tot. altID seen:  201
Tot. altID unseen:   2416
```

The `map_OBOterm_between_release` subroutine returns a list of two anonymous references. The first is an anonymous scalar storing the annotations file in the same format of the input file but with the *obsolete* ontology terms substituted with the *updated* ones. The second reference is an anonymous scalar containing some basic statistics, such as the total unique number of ontology terms (of the old release) and the total number of mapped and unmapped *altID* ontology terms. In addition, all the found pairs `alt_id - id` are returned. In the example run above the anonymous hash is addressed in the output file `data/chicken.goa.mapped.txt` whereas the stats are printed on the shell.

The difference between the *old* and the *mapped* file can be easily displayed by using the `diff` command (in a Linux environment):

```
$ cd data && gunzip -k goa_chicken.gaf.128.gz
$ diff goa_chicken.gaf.128 chicken.goa.mapped.txt > go.ann.diff
```

To give an example, below we show the first `23` lines of the file `go.ann.diff`:

```
75c75
< UniProtKB A0AVX7   TESC     GO:0072661  GO_REF:0000024 ISS   UniProtKB:Q96BS2  P ␣
↪Calcineurin B homologous protein 3  CHP3_CHICK|TESC|CHP3 protein  taxon:9031 ␣
↪20120627 UniProt
```

(continues on next page)

(continued from previous page)

```
---
> UniProtKB A0AVX7   TESC     GO:0072659  GO_REF:0000024 ISS   UniProtKB:Q96BS2  P ␣
↪Calcineurin B homologous protein 3  CHP3_CHICK|TESC|CHP3 protein  taxon:9031 ␣
↪20120627 UniProt
159c159
< UniProtKB A1DYI3   Wnt3     GO:0005578  GO_REF:0000040 IEA   UniProtKB-SubCell:SL-
↪0111  C  Protein Wnt A1DYI3_CHICK|Wnt3|WNT3  protein  taxon:9031  20160507 UniProt
---
> UniProtKB A1DYI3   Wnt3     GO:0031012  GO_REF:0000040 IEA   UniProtKB-SubCell:SL-
↪0111  C  Protein Wnt A1DYI3_CHICK|Wnt3|WNT3  protein  taxon:9031  20160507 UniProt
234,235c234,235
< UniProtKB A1KXM5   SPERT    GO:0016023  GO_REF:0000019 IEA  ␣
↪Ensembl:ENSMUSP00000127439 C  Spermatid-associated protein  SPERT_CHICK|SPERT␣
↪protein  taxon:9031  20160507 Ensembl
< UniProtKB A1XGV6   TNFRSF19    GO:0004872  GO_REF:0000033 IBA  ␣
↪PANTHER:PTN000950406 F  Troy-long   A1XGV6_CHICK|TNFRSF19   protein  taxon:9031 ␣
↪20160114 GO_Central
---
> UniProtKB A1KXM5   SPERT    GO:0031410  GO_REF:0000019 IEA  ␣
↪Ensembl:ENSMUSP00000127439 C  Spermatid-associated protein  SPERT_CHICK|SPERT␣
↪protein  taxon:9031  20160507 Ensembl
> UniProtKB A1XGV6   TNFRSF19    GO:0038023  GO_REF:0000033 IBA  ␣
↪PANTHER:PTN000950406 F  Troy-long   A1XGV6_CHICK|TNFRSF19   protein  taxon:9031 ␣
↪20160114 GO_Central
268c268
< UniProtKB A3F962   MBNL2    GO:0044822  GO_REF:0000019 IEA  ␣
↪Ensembl:ENSP00000365861 F  Muscleblind-like 2 isoform 1  A3F962_CHICK|MBNL2  ␣
↪protein  taxon:9031  20160507 Ensembl
---
> UniProtKB A3F962   MBNL2    GO:0003723  GO_REF:0000019 IEA  ␣
↪Ensembl:ENSP00000365861 F  Muscleblind-like 2 isoform 1  A3F962_CHICK|MBNL2  ␣
↪protein  taxon:9031  20160507 Ensembl
286c286
< UniProtKB A4GTP0   A4GTP0      GO:0044822  GO_REF:0000019 IEA  ␣
↪Ensembl:ENSP00000254301 F  Galectin A4GTP0_CHICK   protein  taxon:9031  20160507␣
↪Ensembl
---
> UniProtKB A4GTP0   A4GTP0      GO:0003723  GO_REF:0000019 IEA  ␣
↪Ensembl:ENSP00000254301 F  Galectin A4GTP0_CHICK   protein  taxon:9031  20160507␣
↪Ensembl
321c321
```

4.2 Human Phenotype Ontology (HPO)

Here we show how to use `obogaf::parser` on the HPO obo file and its annotation file. Here we go faster, because the experiments are carried-out in the same way of those shown above with the GO.

4.2.1 Parse the HPO obo file

Here we use `obogaf::parser` to handle the HPO obo file and return some basic statistics. For this example we use the `2019-11-08` HPO obo release.

```perl
#!/usr/bin/perl

## loading obogaf::parser and useful Perl module
use strict;
use warnings;
use File::Path qw(make_path); ## to recursively create directories
use obogaf::parser;

## create folder where storing example data
my $basedir= "data/";
make_path($basedir) unless(-d $basedir);

## download HPO obo file
my $obofile= $basedir."hpo.obo";
my $hpobo= qx{wget --output-document=$obofile http://purl.obolibrary.org/obo/hp.obo};
print "HPO obo file downloaded: done\n\n";

## shrink HPO obo file to a subset of terms
my @terms = qw(HP:0001507 HP:0000008 HP:0002719 HP:0000021 HP:0000023);
my $termsfile= $basedir."hpoterms.txt";
open OUT, "> $termsfile";
foreach my $go (@terms){print OUT "$go\n";}
close OUT;

$res= obo_filter($obofile, $termsfile);
my $newobo= $basedir."hpo-shrunk.obo";
open OUT, ">", $newobo;
print OUT "${$res}";
close OUT;

## extract edges from HPO obo file
my $hpores= obogaf::parser::build_edges($obofile);
my $hpoedges= $basedir."hpo-edges.txt"; ## hpo edges file declared here
open OUT, "> $hpoedges"; ## redirect hpo edges on file
print OUT "${$hpores}"; ## scalar dereferencing
close OUT;
print "build HPO edges: done\n\n";

## compute parents and children list on HPO ontology
my $parlist= $basedir."gobasic-parHPO.txt";
my $pares= obogaf::parser::get_parents_or_children_list($hpoedges, 0,1, "parents");
open FH, "> $parlist";
foreach my $k (sort{$a cmp $b} keys %$pares) { print FH "$k $$pares{$k}\n";} ##␣
↪parents  list
close FH;

my $chdlist= $basedir."gobasic-chdHPO.txt";
my $chdres= obogaf::parser::get_parents_or_children_list($hpoedges, 0,1, "children");
open FH, "> $chdlist";
foreach my $k (sort{$a cmp $b} keys %$chdres) { print FH "$k $$chdres{$k}\n";} ##␣
↪children list
close FH;
print "\nHPO parents/children list: done\n\n";

## make stats on HPO
my ($parentIndex, $childIndex)= (0,1);
my $res= obogaf::parser::make_stat($hpoedges, $parentIndex, $childIndex);
```

(continues on next page)

(continued from previous page)

```
print "$res"; ## print stats on shell

## results printed on the shell
#oboterm <tab> degree <tab> indegree <tab> outdegree
HP:0003110  60 2  58
HP:0012379  45 1  44
HP:0010876  42 1  41
HP:0000708  39 1  38
HP:0011805  39 1  38
HP:0003355  37 1  36
HP:0012531  36 1  35
HP:0030057  34 1  33
HP:0001760  31 1  30
HP:0008069  31 2  29
.
.
~summary stat~
nodes: 14586
edges: 18416
max degree: 60
min degree: 1
median degree: 1.0000
average degree: 1.2626
density: 8.6567e-05
```

4.2.2 Parse the HPO annotation file

Here we use `obogaf::parser` to parse the HPO annotation file (release `2019-11-08`)

```
#!/usr/bin/perl

## loading obogaf::parser and useful Perl module
use strict;
use warnings;
use File::Path qw(make_path); ## to recursively create directories
use obogaf::parser;

## create folder where storing data
my $basedir= "data/";
make_path($basedir) unless(-d $basedir);

## download HPO annotations
my $hpofile= $basedir."hpo.ann.txt"; ## hpo annotation file declared here
my $hpoann= qx{wget --output-document=$hpofile http://compbio.charite.de/jenkins/job/
↪hpo.annotations.monthly/lastStableBuild/artifact/annotation/ALL_SOURCES_ALL_
↪FREQUENCIES_genes_to_phenotype.txt};

## extract HPO annotations
my ($geneindex, $classindex)= (1,3);
my ($res, $stat)= obogaf::parser::gene2biofun($hpofile, $geneindex, $classindex);
my $hpout= $basedir."hpo.gene2pheno.txt"; ## annotation adj list stored in a file
open OUT, "> $hpout";
foreach my $k (sort{$a cmp $b} keys %$res) { print OUT "$k $$res{$k}\n";} ##␣
↪dereferencing
close OUT;
```

(continues on next page)

(continued from previous page)

```
print "${$stat}\n";

## results printed on the shell
genes: 4293
ontology terms: 7729
```

Below we show the first 10 lines of the hpo.gene2pheno.txt file, just to give an example of how this file is structured:

```
A2M␣
↪HP:0000006|HP:0000726|HP:0001300|HP:0001425|HP:0002185|HP:0002423|HP:0002511|HP:0410054
A2ML1␣
↪HP:0000006|HP:0000028|HP:0000044|HP:0000179|HP:0000218|HP:0000316|HP:0000325|HP:0000347|HP:0000348
A4GALT HP:0000006|HP:0010970
AAAS␣
↪HP:0000007|HP:0000252|HP:0000407|HP:0000505|HP:0000522|HP:0000612|HP:0000648|HP:0000649|HP:0000830
AAGAB␣
↪HP:0000006|HP:0000982|HP:0001425|HP:0001597|HP:0002894|HP:0003002|HP:0003003|HP:0003584|HP:0005584
AARS1␣
↪HP:0000006|HP:0000007|HP:0000252|HP:0000348|HP:0000407|HP:0000494|HP:0000504|HP:0000508|HP:0000546
AARS2␣
↪HP:0000007|HP:0000639|HP:0000716|HP:0000726|HP:0001251|HP:0001257|HP:0001260|HP:0001272|HP:0001332
AASS␣
↪HP:0000007|HP:0000119|HP:0000736|HP:0000750|HP:0000752|HP:0001083|HP:0001249|HP:0001250|HP:0001252
ABAT␣
↪HP:0000007|HP:0000098|HP:0000278|HP:0000494|HP:0001250|HP:0001254|HP:0001263|HP:0001274|HP:0001321
ABCA1␣
↪HP:0000006|HP:0000007|HP:0000505|HP:0000622|HP:0000656|HP:0000958|HP:0000991|HP:0001265|HP:0001349

... to be continued ...
```

4.2.3 Map HPO terms between releases

Here we use obogaf::parser to map the HPO terms of an *old* release (2018-09-03) toward a *new* ones (2019-11-08).

```
#!/usr/bin/perl

## loading obogaf::parser and useful Perl module
use strict;
use warnings;
use File::Path qw(make_path); ## to recursively create directories
use obogaf::parser;

## create folder where storing data
my $basedir= "data/";
make_path($basedir) unless(-d $basedir);

## download HPO obo file
my $obofile= $basedir."hpo.obo";
my $hpobo= qx{wget --output-document=$obofile http://purl.obolibrary.org/obo/hp.obo};

## download HPO old annotation file
my $hpofileOld= $basedir."hpo.ann.old.txt"; ## goa annotation file declared here
my $hpold= qx{wget --output-document=$hpofileOld http://compbio.charite.de/jenkins/
↪job/hpo.annotations.monthly/139/artifact/annotation/ALL_SOURCES_ALL_FREQUENCIES_
↪genes_to_phenotype.txt};
```

(continues on next page)

(continued from previous page)

```
## map HPO terms between releases
my $classindex= 3;
my ($res, $stat)= obogaf::parser::map_OBOterm_between_release($obofile, $hpofileOld,
↪$classindex);
my $mapfile= $basedir."hpo.ann.mapped.txt";
open OUT, "> $mapfile"; ## mapped annotation stored in a file
print OUT "${$res}";
close OUT;
print "${$stat}";

#alt-id <tab> id
HP:0000487  HP:0000486
HP:0000547  HP:0000510
HP:0000655  HP:0007773
HP:0000833  HP:0001952
HP:0001226  HP:0006121
HP:0001322  HP:0006872
HP:0001472  HP:0001426
HP:0001862  HP:0006121
HP:0002271  HP:0012332
HP:0002281  HP:0002282
HP:0002459  HP:0012332
HP:0003464  HP:0003107
HP:0003490  HP:0003150
HP:0005130  HP:0001723
HP:0005364  HP:0004429
HP:0005901  HP:0002754
HP:0006830  HP:0001319
HP:0007314  HP:0002282
HP:0007519  HP:0007485
HP:0007713  HP:0010920
HP:0007758  HP:0000505
HP:0007868  HP:0000608
HP:0007893  HP:0000546
HP:0008012  HP:0000545
HP:0008024  HP:0100018
HP:0008230  HP:0040171
HP:0010700  HP:0000518
HP:0011146  HP:0002384
HP:0012201  HP:0008151
HP:0040290  HP:0003011
HP:0045016  HP:0003455

Tot. ontology terms: 6789
Tot. altID: 3635
Tot. altID seen:  31
Tot. altID unseen:   3604
```

By running the `diff` command between the *old* file (`hpo.ann.old.txt`) and the *mapped* one (`hpo.ann.mapped.txt`) and redirecting the results on a output file (e.g.: `diff hpo.ann.old.txt hpo.ann.mapped.txt > hpo.ann.diff`) we can easily visualize the changed HPO terms between the two release. Below we show just some few lines of `hpo.ann.diff` to give an example:

```
1148c1148
< 51  ACOX1 Tapetoretinal degeneration HP:0000547
```

(continues on next page)

(continued from previous page)

```
---
> 51  ACOX1 Tapetoretinal degeneration HP:0000510
3423c3423
< 190 NR0B1 Decreased testosterone in males  HP:0008230
---
> 190 NR0B1 Decreased testosterone in males  HP:0040171
4041c4041
< 212 ALAS2 Glucose intolerance  HP:0000833
---
> 212 ALAS2 Glucose intolerance  HP:0001952
5049c5049
< 8481   OFD1  Gray matter heterotopias   HP:0002281
---
> 8481   OFD1  Gray matter heterotopias   HP:0002282
6597c6597
< 57724  EPG5  White matter neuronal heterotopia   HP:0007314
---
> 57724  EPG5  White matter neuronal heterotopia   HP:0002282
7244c7244
< 429 ASCL1 Dysautonomia   HP:0002459
---
> 429 ASCL1 Dysautonomia   HP:0012332
7246c7246
```

CHAPTER 5

Scripts

Example scripts on how to apply `obogaf-parser` module respectively to the Gene Ontology (*GO script*) and to the Human Phenotype Ontology (*HPO script*)

5.1 GO script

```perl
#!/usr/bin/perl

## loading obogaf::parser and
use strict;
use warnings;
use obogaf::parser qw(:all);

## elapased time
use Time::HiRes qw(time);
my $start= time;

## recursively create directories ([-p] mkdir option in perl does not work)
use File::Path qw(make_path);

## create folder where storing example I/O files
my $basedir= "data/";
make_path($basedir) unless(-d $basedir);

## note: if you want to store data in your home, use File::HomeDir
# use File::HomeDir qw(home);
# my $basedir = File::HomeDir->my_home."/data/";
# mkdir $basedir unless(-e $basedir);

## declare variables
my ($res, $stat, $parentIndex, $childIndex, $geneindex, $classindex, $parlist, $pares,
↪ $chdlist, $chdres);

```

(continues on next page)

(continued from previous page)

```perl
## ~~ GO OBO ~~ ##
## download GO obo file
my $obofile= $basedir."gobasic.obo";
my $gobo= qx{wget --output-document=$obofile http://purl.obolibrary.org/obo/go/go-
↪basic.obo};
print "GO obo file downloaded: done\n\n";

## shrink GO obo file to a subset of terms
my @terms = qw(GO:0000002 GO:0000003 GO:0000018 GO:0000030 GO:0000038);
my $termsfile= $basedir."goterms.txt";
open OUT, "> $termsfile";
foreach my $go (@terms){print OUT "$go\n";}
close OUT;

$res= obo_filter($obofile, $termsfile);
my $newobo= $basedir."go-shrunk.obo";
open OUT, ">", $newobo;
print OUT "${$res}";
close OUT;

## extract edges from GO obo file
my $gores= build_edges($obofile);
my $goedges= $basedir."gobasic-edges.txt"; ## go edges file declared here
open FH, "> $goedges";
print FH "${$gores}"; ## scalar dereferencing
close FH;
print "build GO edges: done\n\n";

## extract GO subontology nodes and relationships
my @domains= qw(biological_process molecular_function cellular_component);
my %aspects= (biological_process => "BP", molecular_function => "MF", cellular_
↪component => "CC");

foreach my $domain (@domains){
    my $outfile= $basedir."gobasic-edges"."$aspects{$domain}".".txt";
    open FH, "> $outfile";
    my $domainres= build_subonto($goedges, $domain);
    print FH "${$domainres}";
    close FH;
}
print "build edges for each GO subontology: done\n\n";

## make stats on the whole GOobo file
($parentIndex, $childIndex)= (1,2);
$res= make_stat($goedges, $parentIndex, $childIndex);
print "$res";
print "\nGO stats: done\n\n";

## make stats on a GO-BP subontology
my $goedgesbp= $basedir."gobasic-edgesBP.txt";
($parentIndex, $childIndex)= (0,1);
$res= make_stat($goedgesbp, $parentIndex, $childIndex);
print "$res";
print "\nGO BP stats: done\n\n";

## compute parents and children list (whole ontology)
$parlist= $basedir."gobasic-parGO.txt";
```

(continues on next page)

(continued from previous page)

```
$pares= get_parents_or_children_list($goedges, 1,2, "parents");
open FH, "> $parlist";
foreach my $k (sort{$a cmp $b} keys %$pares) { print FH "$k $$pares{$k}\n";} ##␣
↪parents  list
close FH;

$chdlist= $basedir."gobasic-chdGO.txt";
$chdres= get_parents_or_children_list($goedges, 1,2, "children");
open FH, "> $chdlist";
foreach my $k (sort{$a cmp $b} keys %$chdres) { print FH "$k $$chdres{$k}\n";} ##␣
↪children list
close FH;

print "\nGO parents/children list: done\n\n";

## compute parents and children list (GO-BP subontology)
$parlist= $basedir."gobasic-parGO-BP.txt";
$pares= get_parents_or_children_list($goedgesbp, 0,1, "parents");
open FH, "> $parlist";
foreach my $k (sort{$a cmp $b} keys %$pares) { print FH "$k $$pares{$k}\n";} ##␣
↪parents  list
close FH;

$chdlist= $basedir."gobasic-chdGO-BP.txt";
$chdres= get_parents_or_children_list($goedgesbp, 0,1, "children");
open FH, "> $chdlist";
foreach my $k (sort{$a cmp $b} keys %$chdres) { print FH "$k $$chdres{$k}\n";} ##␣
↪children list
close FH;

print "\nGO BP parents/children list: done\n\n";

## ~~ GOA ANNOTATION ~~ ##
## download GO annotation from GOA database (CHICKEN organism)
my $goafile= $basedir."goa_chicken.gaf.gz"; ## goa annotation file declared here
my $goachicken= qx{wget --output-document=$goafile ftp://ftp.ebi.ac.uk/pub/databases/
↪GO/goa/CHICKEN/goa_chicken.gaf.gz};

## extract GO annotation from GOA database (CHICKEN organism)
($geneindex, $classindex)= (1, 4);
($res, $stat)= gene2biofun($goafile, $geneindex, $classindex);
my $goaout= $basedir."chicken.uniprot2go.txt";
open FH, "> $goaout";
foreach my $k (sort{$a cmp $b} keys %$res) { print FH "$k $$res{$k}\n";} ##␣
↪dereferencing
close FH;
print "${$stat}\n";
print "build GOA annotations (CHICKEN): done\n\n";

## ~~ MAP GO TERMS BETWEEN RELEASE ~~ ##
## download old GOA CHICKEN annotation file
my $goafileOld= $basedir."goa_chicken.gaf.128.gz"; ## goa annotation file declared␣
↪here
my $goachickenOld= qx{wget --output-document=$goafileOld ftp://ftp.ebi.ac.uk/pub/
↪databases/GO/goa/old/CHICKEN/goa_chicken.gaf.128.gz};

## map GO terms between release
```

(continues on next page)

(continued from previous page)

```perl
($res, $stat)= map_OBOterm_between_release($obofile, $goafileOld, $classindex);
my $mapfile= $basedir."chicken.goa.mapped.txt";
open FH, "> $mapfile";
print FH "${$res}";
close FH;
print "${$stat}";

## ~~ ELAPSED TIME ~~ ##
print "\n\n";
my $span= time - $start;
$span= sprintf("%.4f", $span);
printf "Elapased Time:\t$span\n";

exit;
```

5.2 HPO script

```perl
#!/usr/bin/perl

## loading obogaf::parser and useful Perl module
use strict;
use warnings;
use obogaf::parser qw(:all);
## elapsed time
use Time::HiRes qw(time);
my $start= time;

## recursively create directories ([-p] mkdir option in perl does not work)
use File::Path qw(make_path);

## create folder where storing example I/O files
my $basedir= "data/";
make_path($basedir) unless(-d $basedir);

## note: if case you want to store data in your home, use File::HomeDir
# use File::HomeDir qw(home);
# my $basedir = File::HomeDir->my_home."/data/";
# mkdir $basedir unless(-e $basedir);

## declare variables
my ($res, $stat, $parentIndex, $childIndex, $geneindex, $classindex, $parlist, $pares,
↪ $chdlist, $chdres);

## ~~ HPO OBO ~~ ##
## download HPO obo file
my $obofile= $basedir."hpo.obo";
my $hpobo= qx{wget --output-document=$obofile http://purl.obolibrary.org/obo/hp.obo};
print "HPO obo file downloaded: done\n\n";

## shrink HPO obo file to a subset of terms
my @terms = qw(HP:0001507 HP:0000008 HP:0002719 HP:0000021 HP:0000023);
my $termsfile= $basedir."hpoterms.txt";
open OUT, "> $termsfile";
foreach my $go (@terms){print OUT "$go\n";}
```

(continues on next page)

(continued from previous page)

```
close OUT;

$res= obo_filter($obofile, $termsfile);
my $newobo= $basedir."hpo-shrunk.obo";
open OUT, ">", $newobo;
print OUT "${$res}";
close OUT;

## extract edges from HPO obo file
my $hpores= build_edges($obofile);
my $hpoedges= $basedir."hpo-edges.txt"; ## hpo edges file declared here
open FH, "> $hpoedges";
print FH "${$hpores}"; ## scalar dereferencing
close FH;
print "build HPO edges: done\n\n";

## make stats on HPO
($parentIndex, $childIndex)= (0,1);
$res= make_stat($hpoedges, $parentIndex, $childIndex);
print "$res";
print "\nHPO stats: done\n\n";

## compute parents and children list on HPO ontology
$parlist= $basedir."gobasic-parHPO.txt";
$pares= get_parents_or_children_list($hpoedges, 0,1, "parents");
open FH, "> $parlist";
foreach my $k (sort{$a cmp $b} keys %$pares) { print FH "$k $$pares{$k}\n";} ##␣
↪parents  list
close FH;

$chdlist= $basedir."gobasic-chdHPO.txt";
$chdres= get_parents_or_children_list($hpoedges, 0,1, "children");
open FH, "> $chdlist";
foreach my $k (sort{$a cmp $b} keys %$chdres) { print FH "$k $$chdres{$k}\n";} ##␣
↪children list
close FH;

print "\nHPO parents/children list: done\n\n";

## ~~ HPO ANNOTATION ~~ ##
## download HPO annotations
my $hpofile= $basedir."hpo.ann.txt"; ## hpo annotation file declared here
my $hpoann= qx{wget --output-document=$hpofile http://compbio.charite.de/jenkins/job/
↪hpo.annotations.monthly/lastStableBuild/artifact/annotation/ALL_SOURCES_ALL_
↪FREQUENCIES_genes_to_phenotype.txt};

## extract HPO annotations
($geneindex, $classindex)= (1,3);
($res, $stat)= gene2biofun($hpofile, $geneindex, $classindex);
my $hpout= $basedir."hpo.gene2pheno.txt";
open FH, "> $hpout";
foreach my $k (sort{$a cmp $b} keys %$res) { print FH "$k $$res{$k}\n";} ##␣
↪dereferencing
close FH;
print "${$stat}\n";
print "build HPO annotations: done\n\n";

```

(continues on next page)

(continued from previous page)

```
## ~~ MAP HPO TERMS BETWEEN RELEASE ~~ ##
## download old HPO annotation file
my $hpofileOld= $basedir."hpo.ann.old.txt"; ## goa annotation file declared here
my $hpold= qx{wget --output-document=$hpofileOld http://compbio.charite.de/jenkins/
↪job/hpo.annotations.monthly/139/artifact/annotation/ALL_SOURCES_ALL_FREQUENCIES_
↪genes_to_phenotype.txt};

## map HPO terms between release
($res, $stat)= map_OBOterm_between_release($obofile, $hpofileOld, 3);
my $mapfile= $basedir."hpo.ann.mapped.txt";
open FH, "> $mapfile";
print FH "${$res}";
close FH;
print "${$stat}";

## ~~ ELAPSED TIME ~~ ##
print "\n\n";
my $span= time - $start;
$span= sprintf("%.4f", $span);
printf "Elapsed Time:\t$span\n";

exit;
```

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

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:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/marconotaro/obogaf-parser/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.

7.1.2 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.

7.1.3 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.

7.1.4 Write Documentation

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

obogaf::parser 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.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/marconotaro/obogaf-parser/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 :)

7.2 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
:::::::::
```

7.3 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 obogaf::parser documentation:

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

7.4 Get Started!

Ready to contribute?

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

   ```
   $ git clone git@github.com:your_name_here/obogaf-parser.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.

7.5 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 `CHANGELOG` file.

CHAPTER 8

Authors

Marco Notaro

CHAPTER 9

History

9.1 obogaf::parser 1.373

9.1.1 New Features

- add the new subroutine obo_filter with test
- improve subroutine get_parents_or_children_list and gene2biofun

9.1.2 Changes

- add case insensitive regex to read obo and gaf file
- improve tutorial

9.2 obogaf::parser 1.272

9.2.1 Changes

- fix CPAN Testers issues

9.3 obogaf::parser 1.271

9.3.1 Changes

-fix CPANTS issues

9.4 obogaf::parser 1.270

9.4.1 New Features

- add the new subroutine get_parents_or_children_list
- add source/destination name of obo terms ID in build_edges and build_subonto subroutines
- fix bug in die condition in build_subonto subroutine

9.4.2 Changes

- adjusted test according to the New Features of build_edges and build_subonto subroutines
- made test for get_parents_or_children_list
- updated reference manual and read the docs documentation

9.5 obogaf::parser 1.016

9.5.1 Changes

- add some degenerate test
- add homepage in Makefile.PL

9.6 obogaf::parser 1.015

9.6.1 Changes

- fix CPAN Testers issues
- improve Makefile.PL

9.7 obogaf::parser 1.014

9.7.1 Changes

- improve obogaf::parser module
- add much more test cases
- add Test::Files dependencies (to fix CPAN issues)
- add regex in MANIFEST.SKIP
- improve Makefile.PL

9.8 obogaf::parser 1.003

9.8.1 Changes

- improve Makefile.PL:
  - add clean attribute
  - add CONFIGURE_REQUIRES
- add regex in MANIFEST.SKIP

9.9 obogaf::parser 1.002

9.9.1 Changes

- fix CPANTS issues

9.10 obogaf::parser 1.001

9.10.1 Changes

- add test
- fix minor bugs
- create documentations on Read the Docs (https://obogaf-parser.readthedocs.io)

9.11 obogaf::parser 0.001

9.11.1 Module Genesis

CHAPTER 10

obogaf::parser License

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```

The End
```