OntoLib Documentation

Release 0.1

Sebastian Koehler, Manuel Holtgrewe

Sep 04, 2017

Installation Tutorial

OntoLib is a modern Java (version 8 and above) library for working with (biological) ontologies. You can easily load an OBO file into an *Ontology* object and work with it in your code. Additionally, there is special support for important biological entities such as HPO and GO that make working with them easier by making important sub ontologies and terms more accessible.

CHAPTER 1

Quick Example

```
HpoOntology hpo;
try (HpoOboParser hpoOboParser = new HpoOboParser(new File("hp.obo"))) {
  hpo = hpoOboParser.parse();
} except (IOException e) {
  System.exit(1);
}

Ontology<HpoTerm, HpoTermRelation> abnormalPhenoSubOntology =
  hpo.getPhenotypicAbnormalitySubOntology();
for (TermId termId : abnormalPhenoSubOntology.getNonObsoleteTermIds()) {
  // ...
}
```

CHAPTER 2

Feedback

The best place to leave feedback, ask questions, and report bugs is the OntoLib Issue Tracker.

Installation

Use Maven Central Binaries

Note: This is the recommended way of installing for normal users.

Simply use the following snippet for your `pom.xml` for using OntoLib modules in your Maven project.

```
<dependencies>
  <dependency>
    <groupId>com.github.phenomics</groupId>
    <artifactId>ontolib-core</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>com.github.phenomics</groupId>
    <artifactId>ontolib-io</artifactId>
    <version>${project.version}</version>
  </dependency>
</dependencies>
```

Install from Source

Note: You only need to install from source if you want to develop OntoLib in Java yourself.

Prequisites

For building OntoLib, you will need

1. Java JDK 8 or higher for compiling OntoLib,
2. Maven 3 for building OntoLib, and
3. Git for getting the sources.

Git Checkout

In this tutorial, we will download the OntoLib sources and build them in `~/Development/ontolib`.

```
~ # mkdir -p ~/Development
~ # cd ~/Development
Development # git clone https://github.com/phenomics/ontolib.git ontolib
Development # cd ontolib
```

Maven Proxy Settings

If you are behind a proxy, you will get problems with Maven downloading dependencies. If you run into problems, make sure to also delete `~/.m2/repository`. Then, execute the following commands to fill `~/.m2/settings.xml`.

```
ontolib # mkdir -p ~/.m2
ontolib # test -f ~/.m2/settings.xml || cat >~/.m2/settings.xml <<END
<settings>
  <proxies>
   <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.example.com</host>
      <port>8080</port>
      <nonProxyHosts>*.example.com</nonProxyHosts>
    </proxy>
  </proxies>
</settings>
END
```

Building

You can build OntoLib using `mvn package`. This will automatically download all dependencies, build OntoLib, and run all tests.

```
ontolib # mvn package
```

In case that you have non-compiling test, you can use the *-DskipTests=true* parameter for skipping them.

```
ontolib # mvn install -DskipTests=true
```

Creating Eclipse Projects

Maven can be used to generate Eclipse projects that can be imported by the Eclipse IDE. This can be done calling `mvn eclipse:eclipse` command after calling `mvn install`:

```
ontolib # mvn install
ontolib # mvn eclipse:eclipse
```

Input / Output

OntoLib provides support for loading OBO files into objects implementing the *Ontology* interface. Currently, there is no generic parsing of OBO files (yet), so you just select one of the supported ontologies (GO, HPO, MPO, or ZPO) and use the specialized parser. For example, for the Gene Ontology:

```
final GoOboParser parser = new GoOboParser(inputFile);
final GoOntology go;
try {
  hpo = parser.parse();
} catch (IOException e) {
  // handle error
}
```

Similarly, for the Human Phenotype Ontology:

```
final HpoOboParser parser = new HpoOboParser(inputFile);
final GoOntology go;
try {
  hpo = parser.parse();
} catch (IOException e) {
  // handle error
}
```

Working with HPO

OntoLib supports you in working with the HPO in the following ways:

- The `HpoOntology` class supports the standard `Ontology` interface. Besides this, the "phenotypic abnormality" sub ontology is extracted on construction and is available through `HpoOntology.getPhenotypicAbnormalitySubOntology()`.
- The classes `HpoFrequencyTermIds`, `HpoModeOfInheritanceTermIds`, and `HpoSubOntologyRootTermIds` provide shortcuts to important/special terms that come in handy when using the HPO.
- OntoLib provides means to parse disease-to-phenotype and disease-to-gene annotations from the HPO project into `HpoDiseaseAnnotation` and `HpoGeneAnnotation` objects.

This section will demonstrate these three features.

The HpoOntology Class

When iterating over all primary (i.e., not alternative) and non-obsolete term IDs, you should use the `getNonObsoleteTermIds()` method for obtaining a `Set` of these ‘‘TermId‘‘s

```
// final HpoOntology hpo = ...
System.err.println("Term IDs in HPO (primary, non-obsolete)");
for (TermId termId : hpo.getNonObsoleteTermIds()) {
  System.err.println(termId);
}
```

You can obtain the correct `HpoTerm` instance for the given `TermId` by consulting the resulting `Map` from calling `getTermMap()`:

```
System.err.println("Term IDs+names in HPO (primary IDs, non-obsolete)");
for (TermId termId : hpo.getNonObsoleteTermIds()) {
  final HpoTerm term = hpo.getTermMap().get(termId);
  System.err.println(termId + "\t" + term.getName());
}
```

The "phenotypic abnormality" sub ontology, can be accessed with ease and then used just like all other `Ontology` objects.

```
// final HpoOntology hpo = ...
final Ontology<HpoTerm, HpoTermRelation> subOntology =
    hpo.getPhenotypicAbnormalitySubOntology();
System.err.println("Term IDs in phenotypic abnormality sub ontology");
for (TermId termId : subOntology.getNonObsoleteTermIds()) {
  System.err.println(termId);
}
```

Shortcuts to Important Term IDs

These can be accessed as follows.

```
System.err.println("ALWAYS_PRESENT\t", HpoFrequencyTermIds.ALWAYS_PRESENT);
System.err.println("FREQUENT\t", HpoFrequencyTermIds.FREQUENT);

System.err.println("X_LINKED_RECESSIVE\t", HpoModeOfInheritanceTermIds.X_LINKED_
↪RECESSIVE);
System.err.println("AUTOSOMAL_DOMINANT\t", HpoModeOfInheritanceTermIds.AUTOSOMAL_
↪DOMINANT);

System.err.println("PHENOTYPIC_ABNORMALITY\t", HpoSubOntologyRootTermIds.PHENOTYPIC_
↪ABNORMALITY);
System.err.println("FREQUENCY\t", HpoSubOntologyRootTermIds.FREQUENCY);
System.err.println("MODE_OF_INHERITANCE\t", HpoSubOntologyRootTermIds.MODE_OF_
↪INHERITANCE);
```

Parsing Annotation Files

You can parse the phenotype-to-disease annotation files as follows.

```
File inputFile = new File("phenotype_annotation.tab");
try {
  HpoDiseaseAnnotationParser parser = new HpoDiseaseAnnotationParser(inputFile);
  while (parser.hasNext()) {
    HpoDiseaseAnnotation anno = parser.next();
    // work with anno
```

```
  }
} except (IOException e) {
  System.err.println("Problem reading from file.");
} except (TermAnnotationException e) {
  System.err.println("Problem parsing file.");
}
```

The phenotype-to-gene annotation file can be parsed as follows.

```
File inputFile = new File("phenotype_annotation.tab");
try {
  HpoDiseaseAnnotationParser parser = new HpoDiseaseAnnotationParser(inputFile);
  while (parser.hasNext()) {
    HpoDiseaseAnnotation anno = parser.next();
    // ...
  }
} except (IOException e) {
  System.err.println("Problem reading from file.");
} except (TermAnnotationException e) {
  System.err.println("Problem parsing file.");
}
```

Querying with Term Similarity

OntoLib provides multiple routines for computing the similarity of terms, given an ontology. For the "classic" similarity measures, the *Similarity* interface provides the corresponding interface definition.

Here is how to compute the Jaccard similarity between two sets of terms.

```
// final HpoOntology hpo = ...
JaccardSimilarity<HpoTerm, HpoTermRelation> similarity =
    new JaccardSimilarity<>(hpo);
// List<TermId> queryTerms = ...
// List<TermId> dbTerms = ...
double score = similarity.computeScore(queryTerms, dbTerms);
```

The Resnik similarity is a bit more complicated as it requires the precomputation of the information content.

```
// final ArrayList<HpoDiseaseAnnotation> diseaseAnnotations = ...
InformationContentComputation<HpoTerm, HpoTermRelation> computation =
    new InformationContentComputation<>(hpo);
Map<TermId, Collection<String>> termLabels =
    TermAnnotations.constructTermAnnotationToLabelsMap(hpo, diseaseAnnotations);
Map<TermId, Double> informationContent =
    computation.computeInformationContent(termLabels);
PairwiseResnikSimilarity<VegetableTerm, VegetableTermRelation> pairwise =
    new PairwiseResnikSimilarity<>(hpo, informationContent);
ResnikSimilarity<HpoTerm, HpoTermRelation> similarity =
    new ResnikSimilarity<>(pairwise, /* symmetric = */true);
// List<TermId> queryTerms = ...
// List<TermId> dbTerms = ...
double score = similarity.computeScore(queryTerms, dbTerms);
```

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/phenomics/ontolib/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.

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.

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.

Write Documentation

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

OntoLib 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.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/phenomics/ontolib/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 :)

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:

Heading 5
~~~~~~~~~


.. heading_6:

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

Documentation Setup

For building the documentation, you have to install the Python program Sphinx. This is best done in a virtual environment. The following assumes you have a working Python 3 setup.

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

```
$ cd ontolib/manual
$ virtualenv -p python3 .venv
$ source .venv/bin/activate
$ pip install --upgrade -r requirements.txt
```

Use the following for building the documentation. The first two lines is only required for loading the virtualenv. Afterwards, you can always use `make html` for building.

```
$ cd ontolib/manual
$ source .venv/bin/activate
$ make html  # rebuild for changed files only
$ make clean && make html  # force rebuild
```

Get Started!

Ready to contribute? First, create your Java/Documentation development setup as described in *install_from_source/doc_setup*.

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

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

3. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making your changes, make sure that the build runs through. For Java:

   $ mvn package

   For documentation:

   $ make clean && make html

6. 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.

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.md` file.

Authors

- Sebastian Bauer
- Peter N. Robinson
- Sebastian Koehler

- Manuel Holtgrewe

Changelog

v0.3

- `xref` tags are now parsed and their content is available in `Term`. Added appropriate classes for representation.
- Added `Ontology.getParent()`.
- Removed `JaccardIcWeightedSimilarity`, `JiangSimilarity`, `LinSimilarity`, supporting code and tests.
- Refactoring the code for object score I/O into `ontolib-io` package.
- Adding support for score distribution reading and writing to H2 database files.
- `Ontology.getAncestorTermIds()` now also resolves alternative term IDs.
- Fixing dependency on slf4j components in `ontolib-core` and `ontolib-io`.
- Adding `getPrimaryTermId()` in `Ontology`.

v0.2

- Making date parser for HPO annotation files more robust. It works now for positive and negative associations.
- Small bug fix in HPO OBO parser.
- Adding `ontolib-cli` package that allows score distribution precomputation from the command line.
- Removed some dead code.
- Added various tests, minor internal refactoring.
- Moved `OntologyTerms` into `ontology.algo` package.

v0.1

- Everything is new.

License

OntoLib is licensed under the BSD Clear 3-Clause License.

```
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
   the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY
THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

How-To: Release on Maven Central

This page describes the steps to release OntoLib on Maven Central.

Read the following first

- http://java.dzone.com/articles/deploy-maven-central
- http://central.sonatype.org/pages/apache-maven.html

Update the `README.rst` file

Change the version in the `README.rst`.

Update the `CHANGELOG.rst` file

- Update the `CHANGELOG.rst` file to reflect the new version.
- Create a new commit with this version.
- Do not create a git tag as this will be done by Maven below.

Update the demo `pom.xml` file

- Ensure that the version OntoLib dependency is set to the non-`SNAPSHOT` version.

Prepare the Release using Maven

```
mvn release:prepare
```

Answer with the default everywhere but use "vMAJOR.MINOR" for giving the tag name, e.g. "v0.15". Eventually, this will update the versions, create a tag for the version and also push the tag to Github.

Perform the Release

```
mvn release:perform
```

Create the release and push it to Maven central/Sonatype.

Releasing the Deployment

Read this:

- http://central.sonatype.org/pages/releasing-the-deployment.html

The publisher backend to Maven Central is here:

- https://oss.sonatype.org/

Update README CHANGELOG

Open README.md and CHANGELOG.md and adjust the files to include the header for the next `SNAPSHOT` version.

Update the demo `pom.xml` file

- Ensure that the dependencies to OntoLib version is set to the non-`SNAPSHOT` version.

Maven comments

- `mvn versions:set` is useful for bumping versions