GO4J

A Set of API used to manipulate Gene Ontology vocabulary

A user manual

By

Guoqing Zhan[1]

Qingming Lu[1]

Qiang Lu[1]

Yixue Li[2]

[1] *Hubei Bioinformatics and Molecular Imaging Key Laboratory，*

Huazhong University of Science and Technology

Wuhan, Hubei 430074, China

[2] *Shanghai Center for Bioinformation Technology*

100 Qinzhou Road, Shanghai 200235,China

Content

1. WHAT IS IT? 3

2. OBTAINING GO4J AND GO 3

2.1. GO4J 3

2.2. GENE ONTOLOGY 3

3. INSTALLATION 4

3.1. REQUIREMENTS 4

3.2. COMPILING 4

4. CONTACT EMAIL 5

5. DISCLAIMER 5

6. EXAMPLES 5

6.1 PARSEREXAMPLE → LOADING GO DEFINITION FILE 5

6.2 GRAPHMODELEXAMPLE → RETRIEVE PATHWAYS BETWEEN GO IDS 6

6.3 GUIEXAMPLE → VISUALIZING GO TERMS 7

Figure 1 structure of GO4J release 3

Figure 2 Tree Graph View 9

Figure 3 Tree View 10

Figure 4 Graph View 11

1. WHAT IS IT?

The intention of this project is to provide a set of Java API for GO (Gene Ontology). It can be used or modified freely under LGPL license.

GO4J has the following modules:

- ✧ GO definition parser: supporting 4 formats, OBO, OBO-XML, RDF-XML and OWL.
- ✧ Graph Model: containing all go ids in a directed graph.
- ✧ Semantic Similarity: evaluating the semantic similarity between two GO ids by optional algorithms.
- ✧ Visualization: visualizing GO pathways of a set of GO ids by Swing.

2. OBTAINING GO4J AND GO

2.1. GO4J

The Home Page of GO4J can be found at http://bioinformatics.org/GO4J/.
For Windows user, please download go4j.zip, and extract files by WinRAR, WinZIP or other decompress tools.
For Linux/Unix user, please download go4j.tar.gz, and extract files as following:

tar xvfz go4j.tar.gz

The extracted files are organized as Fig 1.

```
go4j/
|--- src
|    |--- edu
|    |    |--- …
|    |--- example
|    |    |---ParserExample.java
|    |    |---GraphModelExample.java
|    |    |---GUIExample.java
|    ||--- *.MF
|--- lib
|    |--- jgraph5.4.jar
|    |--- jgraphaddons-1.0.6.jar
|    |--- jgrapht-0.5.3.jar
|    |--- xalan.jar
|    |--- xercesImpl.jar
|    |--- xmlParserAPIs.jar
|--- doc
|    |--- index.html
|    |--- ...
|--- data
|    |--- gene_ontology.obo.2005-06-01
|--- classes
|    |--- ...
|--- build.xml
|--- user manual.pdf
|--- go4j.jar
|--- go4j_model.jar
|--- go4j_parser.jar
```

Figure 1 structure of GO4J release

2.2. Gene Ontology

GO in OBO format

Latest version: http://www.geneontology.org/ontology/gene_ontology.obo
Or Monthly version: ftp://ftp.geneontology.org/pub/go/ontology-archive.

GO in XML or OWL:

http://archive.godatabase.org/latest/

Or, ftp://ftp.geneontology.org/pub/godatabase/archive/latest-full

Attention:

- **An sample GO definition in OBO format is included in the *data* directory of GO4J.**
- **XML and OWL have daily and monthly version.**
- **XML and OWL file should be as:**
  go_<YYYYMM>-termdb.obo-xml.gz
  go_<YYYYMM>-termdb.rdf-xml.gz
  go_<YYYYMM>-termdb.owl.gz
  where <YYYYMM> is the release data of the latest version, such as 200506.

3. INSTALLATION

3.1. Requirements

- jgraph-5.4-java1.4 and jgraphaddons-1.0.6, can be obtained from http://www.jgraph.com.
- jgrapht-0.5.3, can be obtained from http://jgrapht.sourceforge.net/.
- Xerces2 or above, and Xalan2 or above, can be obtained from http://xml.apache.org.
- Ant1.5.x or above, can be obtained from http://ant.apache.org/

Java 1.4.x or above is necessary.

Attention:

- **Ant is not necessary, but it is helpful to compile the source codes.**
- **Three manifest files are provides in *src* directory of GO4J, Please don't modify them if you are not familiar with *jar* and the file format of manifest.**
- **All necessary libraries have been included in the GO4J package.**
- **The newest libraries from raw websites may not work well in GO4J!**

3.2. Compiling

The necessary subdirectories and files for compiling are:

- src contains the source codes of GO4J and examples.
- lib contains necessary libraries.
- build.xml can be used to compile the source codes.

There are 2 methods to compile: by *ant* and by *javac*.

3.2.1. By *ant*

If you have installed *ant*, please change current directory into *go4j*, then typing:

```
ant
```

3.2.2. By *javac*

Otherwise, the GO4J can be complied by *javac*.

On windows

```
cd src
javac -classpath
"../lib/xercesImpl.jar;../lib/xmlParserAPIs.jar;../lib/xalan.jar;../lib/jgrapht-0.5.3.jar;../lib/jgraph5.4.jar;../lib/jgraphaddons-1.0.6.jar" ./edu/hust/go/term/*.java ./edu/hust/go/model/*.java ./edu/hust/go/gui/*.java ./example/*.java
```

On Unix

```
javac -classpath
"../lib/xercesImpl.jar:../lib/xmlParserAPIs.jar:../lib/xalan.jar:../lib/jgrapht-0.5.3.jar:../lib/jgraph5.4.jar:../lib/jgraphaddons-1.0.6.jar" ./edu/hust/go/term/*.java ./edu/hust/go/model/*.java ./edu/hust/go/gui/*.java ./example/*.java
```

Three jar files are generated by *jar*: go4j.jar, go4j_parser.jar, go4j_model.jar. The only difference in the three jar files is their main class.

jar -cmf visual.MF ../go4j.jar edu example
jar -cmf parser.MF ../go4j_parser.jar edu example
jar -cmf model.MF ../go4j_model.jar edu example
cd ..

4. CONTACT EMAIL

Any questions about GO4J distribution, please mailto
hm_zhang101@yahoo.com.cn or luqiang@mail.hust.edu.cn.

5. DISCLAIMER

I am not liable for any damage caused by the use of this program.

6. EXAMPLES

Three example applications are provided in GO4J:

- ✧ ParserExample shows how to load the GO definition into memory by GO parser.
- ✧ GraphModelExample shows how to query the pathways between GO ids and evaluate their semantic similarity.
- ✧ GUIExample shows how to visualize the pathways.

6.1 ParserExample → Loading GO definition file

Usage Example
java –Xmx128M -jar go4j_parser.jar ./data/gene_ontology.obo.2005-06-01 OBO

Attention:

If the GO definition is not in OBO, replace the latest parameter *OBO by their format name*

- ✧ **For OBO-XML, replace** *OBO* **by** *OBO-XML***.**
- ✧ **For RDF-XML, replace** *OBO* **by** *RDF-XML***.**
- ✧ **For OWL, replace** *OBO* **by** *OWL***.**

The first step is to load the GO definition into the memory.

```
// load GO definition
HashMap termMap = GoGraphModel.loadGoDefinition(target, goDefType);
```

The operation of HashMap loadGoDefinition(String definition, byte defType) is as follows:

```
public static HashMap loadGoDefinition(String definition, byte defType)throws    GoException{
        GoParser parser = null;
        HashMap goMap = null;
        try {
            switch (defType) {
                case GoParser.OBO:
                    parser = new OboGoParser(definition);
                    break;
                case GoParser.OBOXML:
                    parser = new OboXmlGoParser(definition);
                    break;
                case GoParser.RDFXML:
                    parser = new RdfXmlGoParser(definition);
                    break;
                case GoParser.OWL:
                    parser = new OwlGoParser(definition);
                    break;
                default:
                    ;
            }
            goMap = parser.getTermMap();
        }
        catch (Exception e) {goMap = null;}
        if(parser==null || goMap==null || goMap.size()==0){
            GoException exe = new GoException(definition, defType);
            throw exe;
        }
        return goMap;
    }
```

All GO ids are held in the goMap. A single go term can be retrieved by the follows:

```
GO_term term=(GO_term)goMap.get("GO: 0015422");
```

6.2 GraphModelExample → retrieving pathways between GO ids

Usage Example
Retrieving pathways between GO:0015422 and its root
java –Xmx128M -jar go4j_model.jar ./data/gene_ontology.obo.2005-06-01 -r 15422

Retrieving common pathways between GO:0015422 and GO:0015423
On Windows:
java –Xmx128M -jar go4j_model.jar ./data/gene_ontology.obo.2005-06-01 -p 15422 15423

Retrieving the semantic similarity between GO:0015422 and GO:0015423
On Windows:
java –Xmx128M -jar go4j_model.jar ./data/gene_ontology.obo.2005-06-01 -s 15422 15423

This example can execute the following functions:
query the pathways between one GO id and its corresponding root or between two GO ids, or evaluate the semantic similarity between two GO ids.

Attention:

Since the contents in the four GO formats (OBO, OBO-XML, RDF-XML and OWL) are same, the go definition in OBO is the only acceptable format, while the other three XML formats are not supported.

In the evaluation of semantic similarity of GO ids, the CAO's method is used, the other four methods, such as LIN, RESNIK,ZZL and oldZZL are not supported too. In fact, all methods are supported in the GO4J.

```
…
// load GO definition
goMap = GoGraphModel.loadGoDefinition(args[0], GoParser.OBO);
…
//transform the non-standard go id, such as 15423, 0015423, GO:15423, into the standard GO id,
//and get the go term instance of edu.hust.go.term.GO_term
term[i-2] = transForm(args[i], goMap);
…
//load all go terms into a graph structure
GoGraphModel graphModel = new GoGraphModel(goMap);
```

Then, the pathways between one GO id and its corresponding root can be retrieved:

```
//Retrieve the pathways
Object[][] objs = graphModel.getPathsOfNode2Root(term);
//show the pathways
System.out.print("The pathways between " + term.getId() + " and the root is:");
for (int i = 0; i < objs.length; i++) {
    System.out.print("\nPath "+(i+1)+":\t");
    for (int j = 0; j < objs[i].length; j++) {
        System.out.print( ( (GO_term) objs[i][j]).getId() + " ");
    }
}
```

the pathways between two GO ids can be retrieved:

```
//Retrieve the common pathways
HashMap map = graphModel.getPathsBetweenNodes(term1, term2);
//Retrieve the common ancestor terms of the two terms
HashSet commonSet = (HashSet) map.keySet().toArray()[0];
//show the common ancestor terms of the two terms
Object[] commomTerms = commonSet.toArray();
System.out.println("The common ancestors between " + term1.getId() + " and " +term2.getId() + " are:\n\t");
for (int i = 0; i < commomTerms.length; i++) {
    System.out.print( ( ( (GO_term) commomTerms[i])).getId() + " ");
}
//show the pathways
System.out.println("\nthe commonPathways are: ");
Object[][] values = (Object[][]) map.get(commonSet);
for (int i = 0; i < values.length; i++) {
      System.out.print("\nPath "+(i+1)+":\t");
      for (int j = 0; j < values[i].length; j++) {
          System.out.print( ( (GO_term) values[i][j]).getId() + " ");
    }
}
```

the semantic similarity between two GO ids can be evaluated:

```
//Evaluate the semantic similarity between GO terms
//the methodType can be Similarity.CAO, Similarity.LIN, Similarity.RESNIK, Similarity.ZZL or Similarity.oldZZL
float value = graphModel.evalSimilarity(term1, term2,methodType);
System.out.println("The semantic similarity between ");
System.out.print(term1.getId() + " " + term2.getId() + " is: "+value);
```

6.3 GUIExample→ visualizing GO terms

Usage Example
java –Xmx128M -jar go4j.jar

A set of GO ids can be visualized by three kinds of view: tree graph, tree or graph. Fig2, Fig3 and Fig4 are examples about these three views. The view can be generated by

```
…
//split go ids by space and comma
String[] ids = parpareMultipleGoId(mIdTextField.getText());
…
//get the child depth in the following pathways.
// -1 means no child included, 0 means all child include, other means real child depth
int childLevel = Integer.parseInt( (String) childLevleComboBox.getSelectedItem());
//create views and log
//viewType can be GraphCreator.JTREEGRAPHVIEW, GraphCreator.JTREEVIEW orGraphCreator. JGRAPHVIEW
Object[] objs = GraphCreator.query(ids, goMap, viewType, childLevel);
// the view
JComponent component = (JComponent) objs[0];
//the log, including invalid GO ids, obsolete GO ids, and valid GO ids.
String log = (String) objs[1];
```

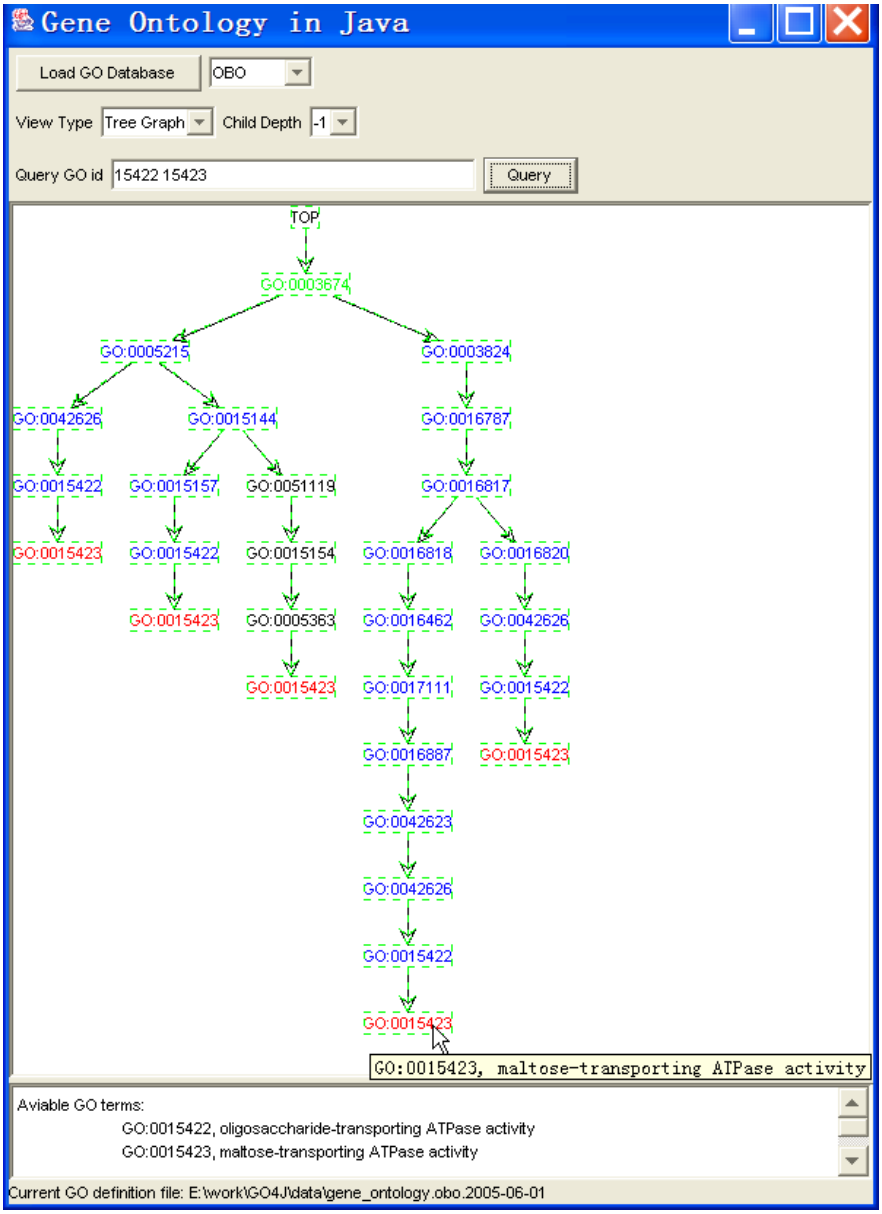

Figure 2 Tree Graph View

The common pathways of GO:0015422 and GO:0015423 in tree graph view. The “top” node is a virtual node. Query ids are 15422 and 15423, whose color is blue; Root id is GO:0003674, whose color is green; the blue nodes are the common ancestors of GO:0015422 and GO:0015423. The black nodes are unique nodes of GO:0015422 or GO:0015423. Putting the mouse on a node, a tooltip about this node will appear.

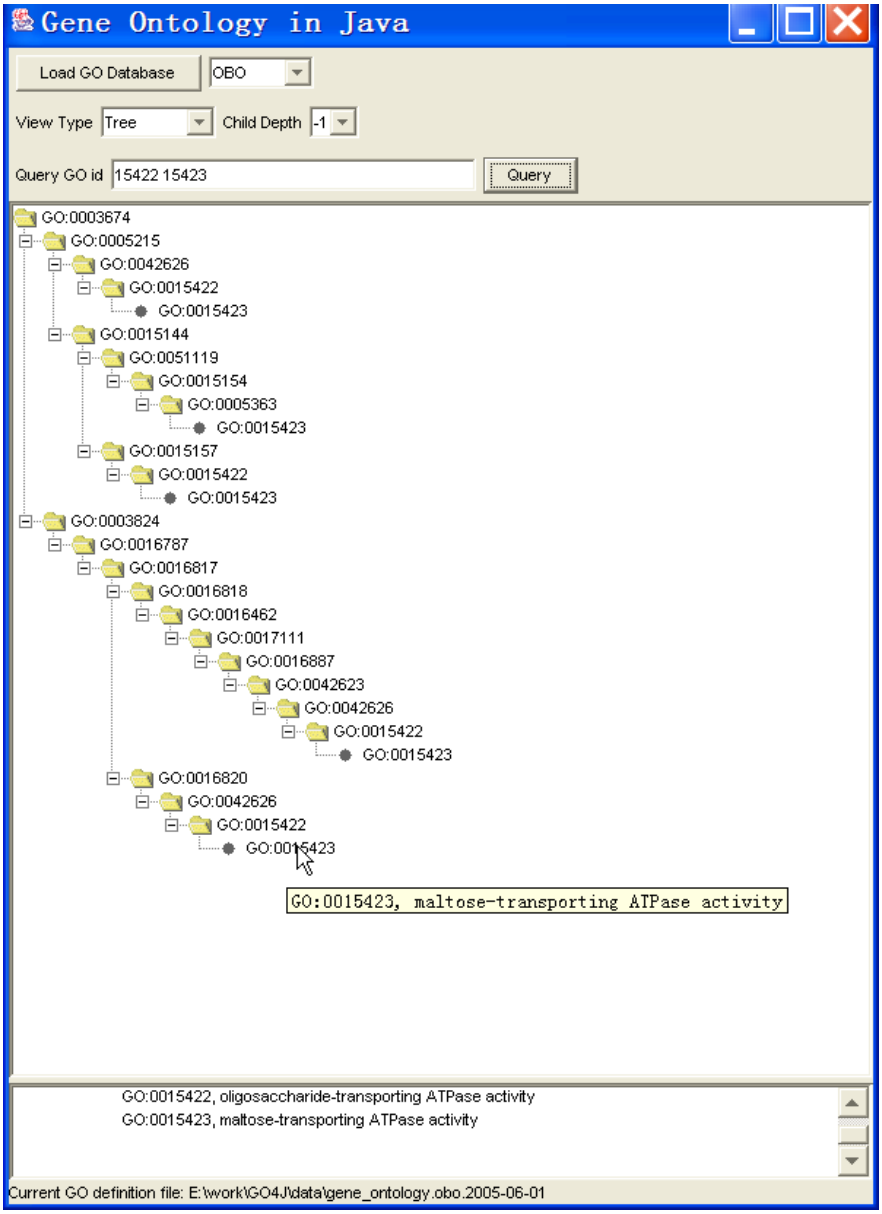

Figure 3 Tree View

The common pathways of GO:0015422 and GO:0015423 in tree view, while the root is GO:0003674. Putting the mouse on a node, a tooltip about this node will appear.

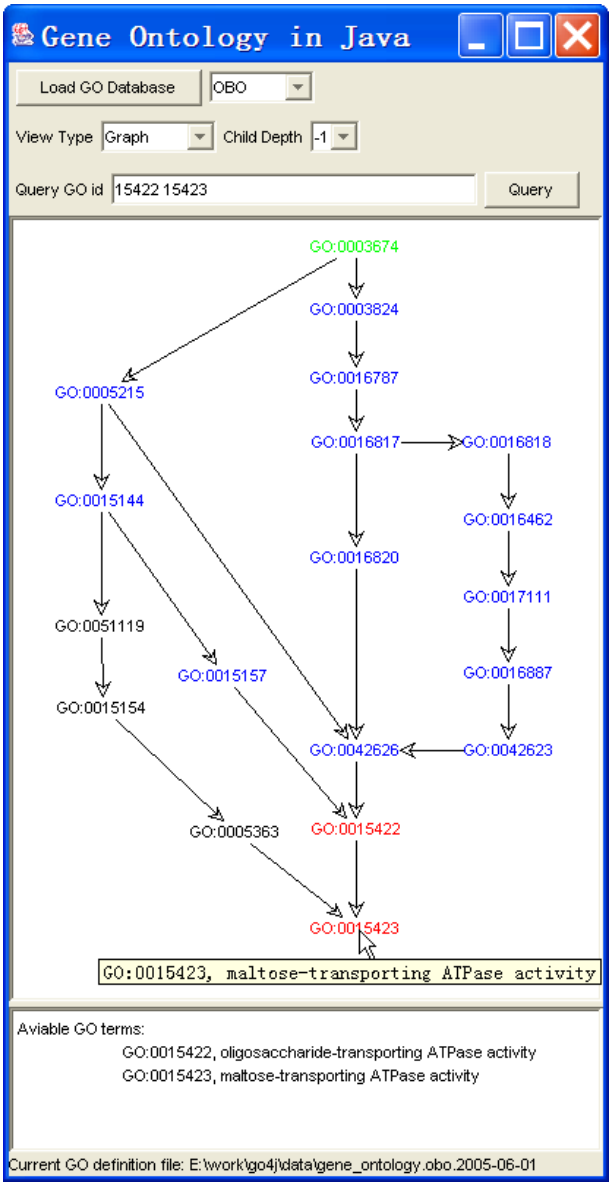

Figure 4 Graph View

The common pathways of GO:0015422 and GO:0015423 in graph view. Query ids are 15422 and 15423, whose color is blue; Root id is GO:0003674, whose color is green. The blue nodes are the common ancestors of GO:0015422 and GO:0015423. The black nodes are unique nodes of GO:0015422 or GO:0015423. Putting the mouse on a node, a tooltip about this node will appear.