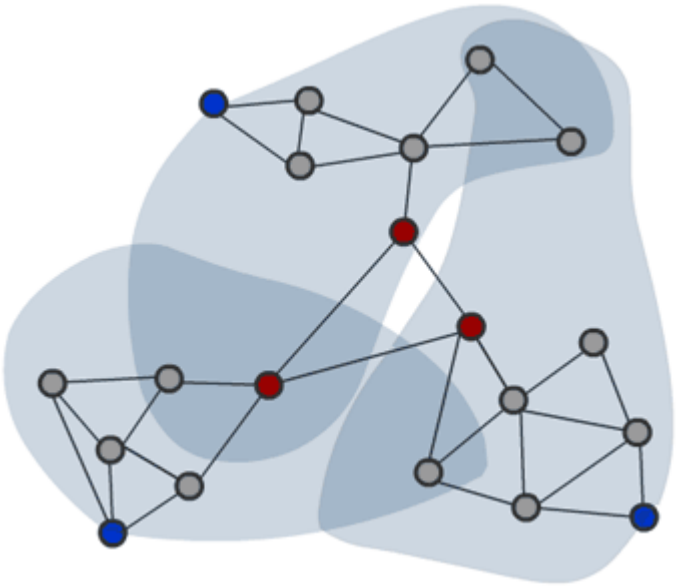

MITANDAO

Social network analyzer

LIBRARY USER GUIDE

Content

What is Mitandao?

Mitandao is an open source software for social network analysis which can be used as a stand alone application or as a library. Mitandao is an extensible application, where you can add your own modules for network analysis. Mitandao library provides useful framework for creating social network analysis application. You can find all important information about Mitandao, as application, library or a framework for developers in our user guides.
Mitandao was developed by group of students of Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, Slovakia.

How to use Mitandao Core Library

Mitandao Core Library is a part of the Mitandao project, but it is possible to use this library separately. The library provides services for loading modules, creating work flow and providing graph analysis.

Creating Mitandao

The access to the Mitandao Library is defined in the Mitandao interface. You can create Mitandao library through the implementation of this interface `MitandaoImpl`.

```
Mitandao mitandao = new MitandaoImpl();
```

Handling modules

The typical analysis consists of four tasks data input, algorithm, filtering data, data output. These four tasks are implemented in four separate modules. Every module contains many implementations of the task e.g. the algorithm module contains various algorithms or input module contains classes for loading graph from various formats.

All classes in modules must implement one from four interfaces `InputModule, AlgorithmModule, FilterModule or OutputModule`. These interfaces extends interface Module and their purpose is to mark what is the module for.

You can obtain the list of available modules from the `ModulesManager`.

```
ModulesManager modulesManager = mitandao.getModulesManager();
List<Module> availableModules = modulesManager.getAvailableModules();
```

Or you can request only modules for specific task.

```
List<InputModule> inputModules = modulesManager.getAvailableInputModules();
```

These methods always return new instances of all modules. For more information about modules see Mitandao Social network analyser developers guide.

Loading modules

For every type of modules a separate List is created. According to module class placement the type of the module is recognized. When the class names list for all modules types is assembled, a class type object and than using java reflexion its instance is created.

```
public  List<InputModule>  getInputModules(){
            List<InputModule> lim=new ArrayList<InputModule>();
            for (int i = 0; i < listInput.size(); i++) {
                  try {
                        Class cl=Class.forName(listInput.get(i).toString());
                        try {
                              InputModule im=(InputModule)cl.newInstance();
                              System.out.println(im.getName());
                              lim.add(im);
```

After this initialization all modules are managed using `ModulesManager` like described previously.

Parameters setting

Every module can contain various parameters, that are marked by annotations. You can obtain the parameters for the specified module.

```
mitandao.getModuleParameters(module);
```

You can set parameters to the module directly by calling its getter methods, or let the library do it for you you only have to fill the parameters HashMap (keys are parameters' names and values are objects, whose values should be filled) and assign it to the module.

```
mitandao.addModule(module, parameters);
```

or

```
mitandao.setModuleParameters(module, parameters);
```

Handling work flow

Every module implements method apply, which takes as its argument graph, performs specific task (e.g data input) and returns loaded or evaluated graph. The returned graph is the input graph to another module. This chain of modules is called work flow. You can add and remove modules to/from work flow through the Mitandao interface.

```
mitandao.addModule(module);
mitandao.addModule(module, position);
mitandao.removeModule(position);
mitandao.removeModule(module);
```

You can replace module on the specified position by combining these two methods.

```
mitandao.removeModule(position);
mitandao.addModule(newModule, position);
```

After every analysis the work flow is cleared, but you can do it also yourself.

```
mitandao.removeAllModules();
```

Analysis

You can start the analysis by calling the analyse method. The method takes one parameter the graph on which the analysis should be performed. If the parameter is null, new graph will be created. This method returns the analysed graph. The Mitandao library uses the JUNG Graph implementation to store the graph data.

```
Graph analysedGraph = mitandao.analyze(graph);
```

Data storage

The results of the analysis are stored in graph user data under the specified key (usually, this is the qualified name of the module class). Mitandao library contains two data holder `MitandaoVertexLabeller` and `MitandaoEdgeLabeller` all results and data should be placed into one of these holders. The library does not guarantee that no other object is placed into the user data, you should therefore check the type when iterating over user data.

```
Iterator i = graph.getUserDatumKeyIterator();
while (i.hasNext()){
      Object o = i.next();
      if (o instanceof MitandaoVertexLabeller){
            // handle vertices
      } else if (o instanceof MitandaoEdgeLabeller){
            // handle edges
      }
}
```

You can get `MitandaoVertexLabeller` and `MitandaoEdgeLabeller` directly for the specific key.

```
MitandaoVertexLabeller labeller =
                        (MitandaoVertexLabeller) graph.getUserDatum(key);
```

But the preferred way is to obtain it from `MitandaoVertexLabeller` directly, because it is type safe and if the datum under the key does not exist, new `MitandaoVertexLabeller` is created.

```
MitandaoVertexLabeller labeller = MitandaoVertexLabeller.getLabeller(graph,
key);
```

The nodes names are stored under the key

```
MitandaoVertexLabeller.DEFAULT_VERTEX_LABELER_KEY
```

You can access it in a simplified way without the key.

```
MitandaoVertexLabeller labeller = MitandaoVertexLabeller.getLabeller(graph);
```

All data are represented as `Strings`. You can access the data in labeller through method `getLabel`.

```
labeller.getLabel(vertex);
```

Error handling

In the process of analysis it is possible, that some exception occurres. In that case an exception is thrown from the apply method defined in the specific module. If an exception is thrown from the apply method, the process of analysis is not interrupted, only the error object is put to the `AnalysisException`, which is thrown at the end of the process of analysis. This exception contains the instance of the last modified graph so the user can get it from the exception via the method `getGraph()`.

Linking modules

The Mitandao uses dynamical loading of the module classes in order to give you the list of available modules. Therefore it does not search classes on the standard classpath, but on the paths specified in the special XML file named modules_paths.xml. This XML file has to be placed somewhere on the classpath (the easiest way is to put it next to the mitandao.jar file).

The sample XML file looks like this:

```
<modulePaths>
   <modulePathToClass>../MITANDAO/bin</modulePathToClass>
   <modulePathToClass>./modules</modulePathToClass>
   <modulePathToClass>file:///home/lula/programs/mitandao/modules</modulePathTo
Class>
</modulePaths>
```

You can use absolute or relative paths in this XML. This path has to lead to the first directory of the package structure. If you use the absolute path, you have to write the protocol prefix file:// at the beginning. If you use relative paths, they are relative to the directory returned by the system property user.dir (by System.getProperty(user.dir)). This is the directory where the application was run.
Notice that even in Windows you can use the / in the relative paths.

Example:
If your directory looks like this:

```
/home/lula/programs/mitandao/modules/sk/fiit/mitandao/modules/MyModule.class
```

And sk.fiit.mitandao.modules.MyModule.class is the full name of your class, than the path specified in the XML would look like this:

```
file:///home/lula/programs/mitandao/modules
```

If you starts your application in the mitandao directory, the relative path would look like this:

```
./modules
```