GRAAu
Genome Rearrangement Algorithm Auditor

User manual

Gustavo Rodrigues Galvão and Zanoni Dias

version 1.0

Contents

Chapter 1

Introduction

1.1 Overview

GRAAu is an audit tool for genome rearrangement algorithms. The audit consists of comparing, for each permutation in the symmetric group, the distance outputted by a given genome rearrangement algorithm with the related rearrangement distance and producing statistics that can be used to analyze the performance of this algorithm.

This document contains information on how to obtain, install and run GRAAu. It was implemented in Java and is distributed under the Apache License, Version 2.0 (http://www.apache.org /licenses/LICENSE-2.0).

1.2 Installation

1.2.1 Requirements

Since GRAAu was implemented in Java, it is expected to run under any Operating System running a Java Runtime Environment (JRE) version 1.6. The user will need the Java Development Kit (JDK) version 1.6 for compiling Java code. For more information, visit the Java SE website http://www.oracle.com/technetwork/java/javase/overview/index.html.

1.2.2 Obtaining the package

The package can be downloaded at http://mirza.ic.unicamp.br:8080/bioinfo/graau.jsf.

1.2.3 Installing the package

To install the package, the user must download the file *graau.zip* (see Section1.2.2) and decompress it under a desired directory. After decompressing the file *graau.zip*, a directory named *graau* will be created, and inside it one will find a directory named *lib* and a directory named *example*.

The *lib* directory contains all the libraries needed to run GRAAu, which include a subset of the Apache Axis2 (http://axis.apache.org/axis2/java/core/) libraries and a library containing the Java implementation of the bzip2 (http://www.bzip.org/) data compressor. The *example* directory contains a class file named *Example.java* which exemplifies how one can audit a rearrangement algorithm with GRAAu.

By doing these steps, the installation of GRAAu is concluded and it is ready to be run.

Chapter 2

Auditing a rearrangement algorithm

2.1 Implementation

To audit a rearrangement algorithm, the user must implement it in a class that implements the *RearrangementAlgorithm* interface and pass an instance of this class as a parameter to *audit* method of *Auditor* class. The *Example* class – which is provided within graau package, see Section 1.2.3 – is a succinct example on how this can be done: It implements the three methods of *RearrangementAlgorithm* interface and implements the main method, where the method *audit* of *Auditor* class is called.

The three methods of *RearrangementAlgorithm* interface are *getName*, *getRearrangement-ModelCode*, and *getDistance*. The first one must return the name of the rearrangement algorithm; the second one, the code of the rearrangement model considered by the rearrangement algorithm; and the last one, the distance of a given permutation with respect to the rearrangement model being considered. The *getDistance* method must contain the implementation of the rearrangement algorithm that the user wants to audit. A table with all rearrangement models covered by GRAAu along with their codes is provided at http://mirza.ic.unicamp.br:8080/bioinfo/graau.jsf.

To instantiate *Auditor* class, the user need to provide the address of the endpoint reference. The value for this address is

```
http://mirza.ic.unicamp.br:8080/axis2/services/BioinfoService
```

After instantiating *Auditor* class, the user must call the method *audit* to initiate the audit. This method receives three parameters: a *RearrangementAlgorithm* object, the number of threads used by *Auditor* object when auditing the rearrangement algorithm (we recommend that this number do not exceed the number of cores of the CPU), and a ticket. This ticket is a 32 byte string that can be generated at http://mirza.ic.unicamp.br:8080/bioinfo/ticket.jsf. The aim of it is to prevent malicious automated programs from executing GRAAu.

2.2 Compilation and execution

The compilation and the execution of the code are done as usual. We only remark that the libraries inside *lib* directory must de included in the classpath. For instance, the *Example* class can be compiled and executed as follows:

```
> cd <directory where graau package was unzipped>
> javac -cp lib/XmlSchema-1.4.3.jar:lib/axiom-api-1.2.8.jar:
lib/axiom-dom-1.2.8.jar:lib/axiom-impl-1.2.8.jar:lib/axis2-adb-1.5.1.jar:
lib/axis2-kernel-1.5.1.jar:lib/axis2-transport-http-1.5.1.jar:
lib/axis2-transport-local-1.5.1.jar:lib/bzip2.jar:lib/commons-codec-1.5.jar:
lib/commons-collections-3.2.1.jar:lib/commons-configuration-1.6.jar:
lib/commons-httpclient-3.1.jar:lib/commons-io-1.4.jar:lib/commons-lang-2.5.jar:
lib/commons-logging-1.1.1.jar:lib/graau.jar:lib/httpcore-4.0.jar:
lib/mail-1.4.jar:lib/neethi-2.0.4.jar:lib/wsdl4j-1.6.2.jar:.
example/Example.java
> java -cp lib/XmlSchema-1.4.3.jar:lib/axiom-api-1.2.8.jar:
lib/axiom-dom-1.2.8.jar:lib/axiom-impl-1.2.8.jar:lib/axis2-adb-1.5.1.jar:
lib/axis2-kernel-1.5.1.jar:lib/axis2-transport-http-1.5.1.jar:
lib/axis2-transport-local-1.5.1.jar:lib/bzip2.jar:lib/commons-codec-1.5.jar:
lib/commons-collections-3.2.1.jar:lib/commons-configuration-1.6.jar:
lib/commons-httpclient-3.1.jar:lib/commons-io-1.4.jar:lib/commons-lang-2.5.jar:
lib/commons-logging-1.1.1.jar:lib/graau.jar:lib/httpcore-4.0.jar:
lib/mail-1.4.jar:lib/neethi-2.0.4.jar:lib/wsdl4j-1.6.2.jar:. example.Example
```

2.3 Tracking the audit progress

The user can track the audit progress at http://mirza.ic.unicamp.br:8080/bioinfo/status.jsf after the audit has been started.

2.4 Important observations

Some important observations about the implementation and execution:

1. The name of the rearrangement algorithm should not exceed 50 characters and it can only contain letters or numbers (spaces or special characters are not allowed).
2. The code of the rearrangement model considered by the rearrangement algorithm is case sensitive, therefore you should copy it exactly from the table.
3. If you intend to perform the audit with more than one thread, notice that the all threads will use the same instance of the class that implements RearrangementAlgorithm interface. Therefore, unless this class is stateless, you will need to be careful to avoid race conditions.

4. A ticket is valid for just 15 minutes, and it becomes invalid after one uses it.

5. To resume the audit of a rearrangement algorithm after it has been interrupted, all you have to do is to restart its execution. There is no need to generate another ticket.

6. After execution starts, a file named "*RearrangementAlgorithmName*.session" is created inside the application root directory (in the example given in Section 2.2, a file named "Example.session" would be created inside *graau* directory). It contains data used for authentication purposes. Do not delete, move, or rename this file because GRAAu uses it to resume the audit of a rearrangement algorithm.

Chapter 3

Viewing and editing audit results

The user can view the audit results at http://mirza.ic.unicamp.br:8080/bioinfo/results.jsf. In this section, the user can also view further information about each rearrangement algorithm audited or being audited by GRAAu: its name, the rearrangement model considered, and its description. The name and the description of an algorithm can be edited; the audit results, and even all the information about an algorithm, can be deleted. To perform any of these operations, the user has to enter a 128-bit password that was recorded in the file "*RearrangementAlgorithm-Name*.session" after the audit started (see Section 2.4).