Reduce And prIoritize SuitEs: A Tutorial

This tutorial will assist the user in acquiring, installing, and using the RAISE package.

1 Acquisition

SVN: If you would like the RAISE sourcecode, you may obtain a checkout from the subversion repository with the following command.

```
svn checkout http://raise.googlecode.com/svn/ <destination-folder>
```

You may alternatively download the jar file from raise.google.com by clicking on the downloads tab.

2 Installation

2.1 Build From Source

Depenencies: There are a few requirements that must be fulfilled before the RAISE package can be built.

- RAISE requires at least java version 1.5.
- The source code comes with an ANT build.xml file, so ANT (ant.apache.org) must be installed in order to build the package using the ant build file.

Compilation: All ANT commands must be executed from the root directory of the project. The root directory contains the 'src', 'lib', 'build', 'doc', and 'data' directories.

To compile the source code type,

```
ant compile
```

After you have built the package, you can execute the RAISE test suite by typing the command,

```
ant testSetCover
```

Install: To use the package from anywhere on your machine, add the full path to the raise/src directory to your classpath environment variable. Also you will need to add the raise/lib/xstream-1.1.3.jar file to your classpath.

2.2 Jar File

If you have downloaded the jar file, you simply have to add the full path to the jar file in your CLASSPATH environment variable. For example, if you have the jar file in your home directory on your linux machine the classpath must include `/usr/home/your_username/raise.jar`.

1	0	0	1	2
0	1	0	1	2
1	0	1	0	2
2	1	1	2	0

Figure 1: An example coverage matrix.

Name	Time
1	4.0
2	1.0
3	2.0
4	3.0

Figure 2: An example time file.

3 Using the Package

3.1 Creating a Test Suite Representation

The RAISE package reduces and prioritizes test suites. A test suite is represented by the SetCover object. In your source code create a new test suite with the line,

```
SetCover sc = new SetCover()
```

Test suites are represented as a combination of coverage matrix and time information files. The coverage matrix files contain a binary matrix whose columns represent tests and rows represent requirements. A 1 at row `i` and column `j` indicates that test `j` covers requirement `i`. The last row and column of each file contains the summary data for that row or column. The number in the last column of each row represents the total number of tests that cover the requirement represented by the row. The number in the last row of each column represents the total number of requirements that the test represented by the column covers. Figure 1 shows an example coverage matrix file. In this example the first column represents a test that covers the first and third requirements, but not the second. The fact that the test covers two requirements is represented in the last row for the column. The first row shows a requirement that is covered by the first and fourth tests, but not the second and third. The summary column shows that the requirement is covered by two tests.

Each matrix file is accompanied by a file that stores the timing information for each test. The first column contains the heading 'Name' and then the index of each test. The second column first has the heading 'Time' and then a time value for each test. In Figure 2, test 1 executes for 4.0 seconds, test 2 executes for 1.0 seconds, test 3 executes for 2.0 seconds, and test 4 executes for 3.0 seconds.

After you have created a matrix file and time file, you can build the SetCover object using a static method that returns a SetCover object that is defined by those files.

```
sc = SetCover.constructSetCoverFromMatrix(String matrixFile, String
                                          timeFile)
```

3.2 Reducing and Prioritizing:

After the SetCover object has been created, there are 4 different algorithms to choose from that perfrom reduction abd prioritization. Each algorithm takes a String paramter that represents the greedy choice metric to be used. This can be `'ratio'`. `'time'`, or `'coverage'`. The Harrold Gupta Soffa algorithm also takes an integer `numberOfLooksAhead`

which is used to break ties in the algorithm. For complete explanations of each algorithm, please consult the SAC 2009 paper in the doc directory.

- Harrold Gupta Soffa
  - `reduceUsingHarroldGuptaSoffa(String metric, int numberOfLooksAhead)`
  - `prioritizeUsingHarroldGuptaSoffa(String metric, int numberOfLooksAhead)`
- Delayed Greedy
  - `reduceUsingDelayedGreedy(String metric)`
  - `prioritizeUsingDelayedGreedy(String metric)`
- Greedy
  - `reduceUsingGreedy(String metric)`
  - `prioritizeUsingGreedy(String metric)`
- 2-Optimal
  - `reduceUsing2Optimal(String metric)`
  - `prioritizeUsing2Optimal(String metric)`

4 Evaluating Results

Currently, the reduction and prioritization algorithms destroy the SetCover objects when they are executed. To compare the results to the original test suite, the values that will be compared must be saved before the reduction or prioritization algorithm is run, or the `restoreSetCover()` class method can be called to restore the SetCover in such a way that the data can be retreived. This restoration does not garauntee that another algorithm may be run. In order to evaluate more reduction and prioritization techniques, the SetCover must be fully reconstructed from the matrix and time files.

Evaluating Reduction: After the SetCover has been reduced, a LinkedHashSet of the new test suite can be obtained using the SetCover class method `getCoverPickSets()`. This collection of tests is garaunteed to cover all of the requirements that were covered by the original test suite. The size of the LinkedHashSet object can be retrieved using `<linkedHashSetInstance>.size()`. This can be compared to the size of the original test suite size which can be obtained (either before the algorithm is run, or after restoring the SetCover) by using `<setCoverInstance>.getTestSubsets().size()`.

Evaluating Prioritization: To evaluate the effectiveness of a prioritization, RAISE uses coverage effectiveness (CE) (See the SAC2009 paper for details). The SetCover class method `<setCoverInstance>.getCE(int[] order)` returns the CE of the prioritized test suite. The int array parameter represents the order of the new test suite. So the integer value at order[0] is the index of the test to be run first, the integer value at index[1] is the test to be run second, and so on. Again, the SetCover must be rebuilt or restored to get the CE of the SetCover after prioritization.