University of Illinois at Chicago

Master's Project

MINing Useful Sequences for ASSIST(MINUSA)

Author:
Kush Shah
UIN:
663858880

Supervisor:
Dr. Mark Grechanik
Secondary Reader:
Dr. Ugo Buy

A report submitted in fulfilment of the requirements for the degree of Master of Science

in Computer Science

Department of Computer Science

Spring 2014

Declaration of Authorship

I, Kush Shah, declare that this report titled, 'MINing Useful Sequences for ASSIST(MINUSA)' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this University.
- Where any part of this report has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the report is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UNIVERSITY OF ILLINOIS AT CHICAGO

Abstract

Mark Grechanik
Department of Computer Science

Master of Science

MINing Useful Sequences for ASSIST(MINUSA)

by Kush Shah

The main aim of the project is to generate meaningful and optimized sequences from the huge number of sequence pools available from the execution traces and the state carving applied on them.

This activity can be divided into various sub tasks:

→ Parsing the JSON format output of ASSIST and generating an input file for the Frequent Closed Sequence Miner(BIDE)[1].

→ Applying Frequent Pattern Miner using different thresholds on the same data and receiving different outputs from BIDE.

→ Finding out the frequent sequences in all order and quantity present in the Run Sequences using Frequent Patter Miner output.

→ Optimize the sequences using heuristics(shortest sequence, ranked shortest sequence,etc.) so as to get the most information out of it.

→ Revert the output back to ASSIST in JSON fromat.

Acknowledgements

I would like to take this opportunity to thank my advisor Dr. Mark Grechanik, without whose guidance, help and support MINUSA would not have been possible. I thank him for his efforts in making the project reach a completion within stipulated period.

I would also like to thank Guru Devanla, for his continuous help, during the planning and implementation phase. It would have been very difficult without his help and vision.

I would also like to extend my sincere thanks to all my colleagues who helped me in one way or the other.

Contents

List of Figures

Abbreviations

JSON	**J**ava **S**cript **O**bject **N**otation
ASSIST	**A**utomatic **S**ynthesi**S** of **I**ntegration **S**oftware **T**ests
AUT	**A**pplication **U**nder **T**est
ATS	**A**pplication **T**est **S**uite
BIDE	**BI** **D**irectional **E**xtension
SAX	**S**imple **A**PI for **X**ML
POM	**P**roject **O**bject **M**odel

Chapter 1

Integration Testing

1.1 Contemporary Softwares

Companies today are embracing *agile software development* to a greater extent for their softwares.

The two main facets of agile technologies are *incremental* and *iterative*. They incorporate the ever changing requirements and aim to incorporate all. The very nature of the development practice being incremental requires the employees to have sprints and/or scrums for the projects. It is basically a short period in time after which the developers and managers meet and incrementally build upon the requirements or even change them if required.

Due to this incremntal and iterative approach, testing plays a very important role in the process. More often the releases are between very small amount of time periods like two weeks or may be less.

This creates a lot of versions with varied requirements and as the software is built incrementally and has smaller and broken up modules, *Integration Testing* is very important and supposed to be be sound and intuitive.

1.2 Integration Testing

Integration testing involves testing components and/or modules in there entirety to check the correctness of the software.

As discussed earlier, integration testing becomes important for agile development methods but it is also important in any software development methods. Integration Testing is a kind of black box testing used to test, as the name suggests the integrity of the software. Shared data and processes are simulated so that components are tested to behave and interact correctly among each other.

At one end of the spectrum of integration testing we can view acceptance testing, which integrates all components of a system and evaluates according to the requirements. Whereas on the other end of the spectrum lies unit testing where a units(functions) belonging logically or structurally together(eg. class) are tested individually and then combined with similar other units to confirm functional correctness using test harnesses.

Having said this, one needs a sound balance between both the approaches as a perfect integration testing would involve testing methodologies of both the extremes. As the system grows larger, it becomes exponentially difficult to test all the combinations possible. Thus it becomes more important for a better integration testing. Using the unit testing paradigm makes the testing very sluggish as it grows and randomly checking components to work together may be insensible because they may or may not interact in the system. Thus, generating good integration tests involves not only a significant effort to compose differents simulation of states of various components but also a good knowledge of the components and their internals and an acute intuitive niche of combining different components.

1.3 Problem

Thus from the above information we can conclude that setting up an Integration Testing mechanism is an overhead and is a burdern financially on the system at the initial phases but several case studies have proved that they increase the quality of a software in the long run.

It has become more important to have integration test suites up and running from a very early stage looking at the Test Driven Development approach in the agile methodologies.

One looses a count and scale of test suites once the project starts to grow at a faster pace as the suites grow exponentially with the increase in number of components becasue of the many combinations possible. There is also a factor of time as in agile development releases are too nearly placed with respect to time. Thus there is an urgent need to make this process as automated as possible.

This project contributes in a very small way to the larger project and research carried out by Dr. Mark Grechanik in the from of Automatic SynthesiS of Integration Software Tests(ASSIST)(explained in the Chapter2).

Chapter 2

Automatic SynthesiS of Integration Software Tests(ASSIST)

2.1 Overview

Automatic SynthesiS of Integration Software Tests(ASSIST) aims to provide a new solution using unit tests and acceptance tests. It fulfills various objectives in order to achieve the aim.

2.1.1 Synthesis of Fewer and Tighter Integration Tests

There is a need to decrease the number of integration tests on synthesis of which they are more effective which also means it has a increased power of finding bugs. This is because as we discussed randomly generating integration tests for all combinations of components increases costs in terms of time and resources. Thus, one of the objective is to produce an acceptable number of integration tests with a high bug finding power.

To reach this goal, the system will use the acceptance and unit tests to get models that can enhance the synthesis of integration tests. The Application Under Test(AUT) is run and execution traces are collected using a runtime monitoring system. It will generate models that describe the relationship between properties of input data for AUT

and frequent interactions between various components, obtained by mining patterns of method invocations in execution traces. *The system reduces this computationally intensive task by pruning the sequences of components that are not tightly integrated assuming that loosely integrated components have lesser possibility of integration bugs.* The above objective is the goal of the project and is explained in detail in Chapter4.

ASSIST will prioritize synthesis of those integration tests which have interactions among components in the AUT like if they exchange more data or if they invoke each other's methods during execution of system with various input data, or execution path in object of one class affect the object of another class. Thus these types of components are considered tighter and are intuitively lead us to integration bugs.

2.1.2 Meaningful Oracles

Integration tests are more meaninful if they have better *oracles* (methods for checking of the AUT have given correct outputs on a particular execution [2]). Generating oracles is one of the most difficult issue of general software testing [2–6]. [7] suggests that as much as 25% of the time is spent in testing oracles by many companies.

We can create an oracle by capturing the state at the very end of the completion of execution of the last method in an AUT but it would be a very huge oracle in terms of size and even the states will be very large to manage and maintain for each integration test. It is kind of misguiding if the unused values do not match, testers may keep finding errors which even donot occur. Thus, it is diffcult to maintain and evolve tests without specific fields and values in an oracle [8].

System overcomes the issue by projecting carved AUT states onto field used in assertions(*assert* statements) of the unit tests for synthesized integration tests. These are the fields that hold data flow and control flow informtion. By using symbolic execution [9–16] we can deal with this objective.

2.1.3 Evaluate

To evaluate the effectiveness of the results Mutation Integration will be used.

2.2 Organization

Below shown and mentioned are the various steps in the ASSIST project.

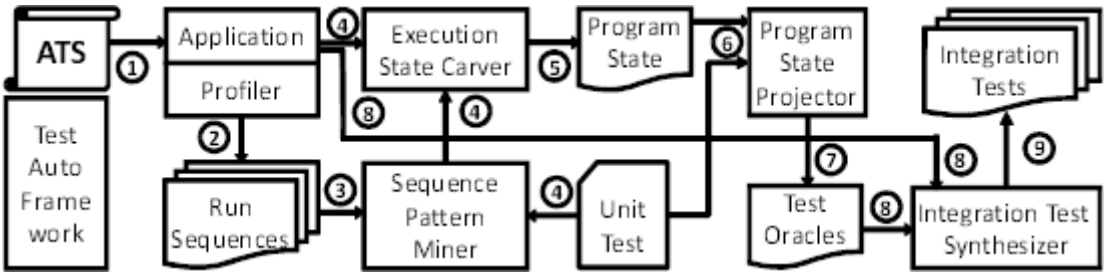

FIGURE 2.1: The architecture and workflow of ASSIST.

1. The Application Test Suite(ATS) comprises of AUT, unit and acceptance tests and input data for these tests.

2. AUT is run using with a profiler which collects *run sequences* (execution traces) which contain method names and class names.

3. Run Sequences are analysed by Sequence Pattern Miner that mines patterns from the various runs.

4. A subset of the most informative run sequences are submitted to the State Carver.

5. State Carver extracts the state of heap memory in the execution of the AUT before the extracted patterns are run.

6. Using unit tests and state of the system Program State Projector determines test oracles.

7. These oracles and source code of AUT are used by Integration Test Synthesizer.

8. Integration tests are outputted.

Chapter 3

Project Requirements

3.1 Requirements

As seen in Chapter 2 - 2.2, this project deals with steps 3 and 4. We can assume them to be the main requirements. I will re-iterate them as follows:

- Run Sequences are analysed by Sequence Pattern Miner that mines patterns from the various runs.
- A subset of the most informative run sequences are submitted to the State Carver.

We will see them on a higher level:

3.1.1 Finding Frequent Sequences

3.1.1.1 Get the files of Run Sequences(in the format of JSON files)

The project receives JSON (a data fromat explained in Chapter-4) files and this becomes the primary input to the project . This file has all the Run Sequences with various information like a unique id, method name , class name, etc. A format is also provided and explained in detail in the Chapter-4. We need to parse and read the files for further use.

3.1.1.2 BIDE Input generation

To find the frequent sequences we use a sequential pattern miner BIDE developed and maintained by University of Illinois at Urbana Champagne. For that we make an input file for the Sequential Pattern Miner(BIDE[1] explained in Chapter- 4). The input file requires a special format as a text file and also some statistics about the data in a spec file, like longest sequence, average sequence etc. which needs to be generated from the run sequences and fed to BIDE.

3.1.2 Mine Informative Sequences

3.1.2.1 Find all possible valid sequences

Use BIDE output file(containing frequent sequences) to mine the Run Sequences and find all the *occurences* of the frequent sequences from the input JSON file. This seems to be a straightforward requirement but it requires careful consideration of many corner cases and with increasing sequence length the problem takes exponential time to solve. We have devised an algorithm to take care of the problem. For eg. {1 2 4 10} is a frequent sequence and to find it out in a following run sequence {1 2 4 10 12 3 5 7 4 2 7 4 8 10} is a daunting task as we need to find all the possible occurences of the frequent sequence not considering its place in the run sequence meaning {1 2 4 10} at places <1 2 3 4>and <1 10 12 14>both are of importance as they are different in order of occuring in a particular run sequence and they both need to be tested as their states are most probably different to each other and thus will provide us with different information when tested.

3.1.2.2 Pruning

Applying the heuristic to get the *shortest distance frequent sequences* or *ranked shortest distance frequent sequences*. This prunes the solution space by a big factor and thus simplifies further situations. Shortest distance is as in the above example {1 2 4 10} at places {1 2 3 4} which is the shortest distance apart from another. Ranked shortest distance is the first n shortest sequences.

3.1.2.3 Output File

Returning the subset of Run Sequences back in a JSON file.

Chapter 4

Tools and Utilities

4.1 JSON(JavaScript Object Notation)

4.1.1 Language Overview

JSON [17] (JavaScript Object Notation) is a lightweight data-interchange format.It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language.JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON has only two structures:

1. A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

1. An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

2. An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).

3. A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.

4. A string is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.

5. A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.

6. Whitespace can be inserted between any pair of tokens. Excepting a few encoding details, that completely describes the language.

4.1.2 json-simple(Parser API)

JSON.simple[18] is a simple Java toolkit for JSON by google. It has following features:

- Full compliance with JSON specification (RFC4627) and reliable (see compliance testing).
- Provides multiple functionalities such as encode, decode/parse and escape JSON text while keeping the library lightweight.
- Flexible, simple and easy to use by reusing Map and List interfaces.
- Supports streaming output of JSON text.
- Stoppable SAX-like interface for streaming input of JSON text (learn more).
- Heap based parser.
- High performance (see performance testing).
- No dependency on external libraries.

- Both of the source code and the binary are JDK1.2 compatible.

4.2 Formats

4.2.1 Input File Format

There will be two files coming in to the project, the sequence file stating the run sequences of the system. Each run system is described by the function calls called in order for that particular run. They are denoted by number here which is mapped in another file with the other details like the method name and class name. It is possible that the same method be called multiple amounts of time in a single run and thus a counter number is associated to uniquely identify the method at that position.

1. Method Trace:

   [{"counter": 0, "class-name": "com.ser.instrument.artifacts.InstrumentClass1", "method-name": "int methodWithPrimitiveInt(int)", "level": 28},
   {"counter": 1, "class-name": "com.ser.instrument.artifacts.InstrumentClass1", "method-name": "int methodWithPrimitiveInt(int)", "level": 29}
   ]

2. Sequence Map File:

   [{"sequence-no":1, "counters" : [1,2, 3, 4, 5, 6, 7, 8, 9, 10]},
   {"sequence-no": 2, "counters" : [5, 7, 8, 10, 13, 15]}
   ]

4.2.2 Output File Format

There will be one output file generated which will have the frequent sequences for each run provided in the input file. This basically is an array for each frequent sequence(here called method-sequence). With each frequent sequence we attach the counters at each run sequence.

Output:

```
[ {"method-sequence" : [ 1, 3, 4 ],
    "occurences" :
        [{ "sequence-no" : 1,
        "counters" : [ 100, 102, 103] },
        { "sequence-no": 2,
        "counters" : [ 100, 102, 105] },
        { "sequence-no" : 3,
        "counters" : [ 500, 501, 510] }
        ]},

{"method-sequence" : [ 8 ,9, 10 ],
    "occurences" :
        [{ "sequence-no" : 8,
        "counters" : [ 100, 102, 103] },
        { "sequence-no": 9,
        "counters" : [ 100, 102, 105] },
        { "sequence-no" : 10,
        "counters" : [ 500, 501, 510] }
        ]}
]
```


4.3 Maven

4.3.1 Overview

Maven[19] is a build tool made available by Apache. There are various uses of this tool as described below:

- Making the build process easy.
- Providing a uniform build system.
- Providing quality project information.
- Providing guidelines for best practices development.

- Allowing transparent migration to new features.

4.3.2 Project Hierarchy

Maven provides a well organized heirarchy for the project separating test and source folders in test and main folders. The snapshot of the project heirarchy is shown in Fig.4.1.

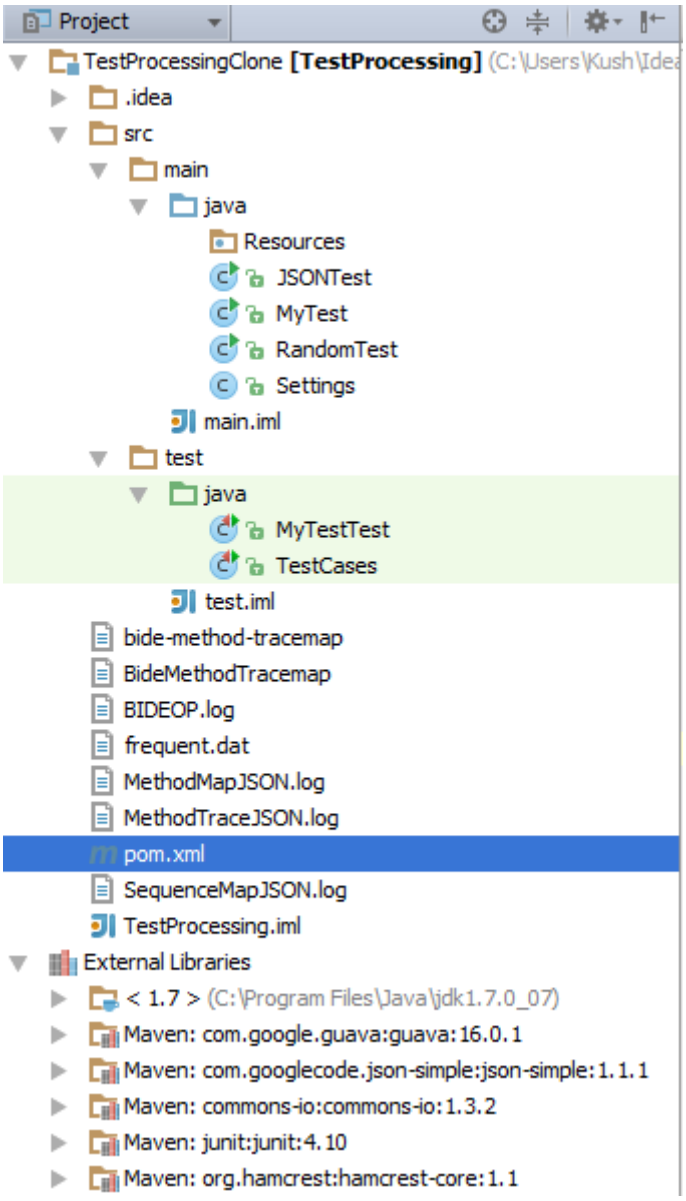

FIGURE 4.1: Project Heirarchy.

4.3.3 Project Object Model

A Project Object Model or POM[20] is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/main/test; and so on.

The main use of the file is to include jar files of external resources like json parser, junit, etc. The build tool fetches the jars from the internet and is thus useful while collaborating as only pom.xml can be shared instead of the physical jar themselves.

Following Fig.4.2 is the pom.xml for the project:

4.4 BIDE (BI-Directional Extension based frequent closed sequence mining)

4.4.1 Overview

The goal of using this tool is to mine frequent patterns of method invocations invoked in execution traces that were collected by the profiler from the AUT with acceptance and get models of interactions among different components.

Frequent patterns are sequences occuring in the dataset more than a predefined and user defined threshold value[21].

For example, a person visits a mall every week and buys some items. He will have a list of items that he bought, suppose each as a row in the dataset. If for assumption, milk and bread appear more than the threshold value, in the rows, they are called as Frequent Sequential Pattern.

{1 2 4 10} is a frequent sequence in a following run sequence {1 2 4 10 12 3 5 7 4 2 7 4 8 10} This sequence is closed and maximal because they are the biggest sequences that are not present in larger and equally or higher frequent sequences.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    <modelVersion>4.0.0</modelVersion>

    <groupId>groupId</groupId>
    <artifactId>TestProcessing</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1.1</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>16.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-io</artifactId>
            <version>1.3.2</version>
        </dependency>
    </dependencies>

</project>
```

FIGURE 4.2: pom.xml.

Thus, we need a miner developed by academicians from University of Illinois at Urbana-Champagne [1] It is a very easy to use executable which can be used by accessing the operating system process for running it.

Let us go through the details of using BIDE:

4.4.1.1 Input parameters

1st argument: The specification file of the dataset

2nd argument: Relative support in decimal

Usage example: bide $bide_input$.spec 0.0002

Where bide is the executable file name, *bide_input*.spec is the specification file of the sequence dataset being mined, 0.0002 is the relative support. As to the dataset file format, see section 4.4.1.4.

4.4.1.2 Output

The discovered frequent sequences are printed into a file called "frequent.dat".
Each line in the result file, "frequent.dat", contains a frequent sequence in the form:
event1 event2 ... eventn : absolute support
Here is an example:
6 24 748 : 66

4.4.1.3 Specification file format

The first line is the dataset file name.
The second line is the number of unique items.
The third line is the number of sequences.
The fourth line is the maximal length of a sequence.
The fifth line is the average length of a sequence.

4.4.1.4 Dataset file format

Usually a sequence database consists of a series of sequences (strictly speaking, here a sequence is a string in the current implementation). Each line represents a sequence and ends with -1, and the entire dataset ends with -2.
Here is a sample sequence:
In our case each number is an item in run sequence more precisely a function invocation.
And each line represents a different run sequence.
For eg.
1 4 35 90 -1
1 2 3 54 77 -1
-2

4.4.2 Other Tools

1. Java Standard Edition Development Kit 7 for coding.
2. IntelliJ (Intgrated Development Environment).
3. HP-ENVY 4 Laptop for all the experiments.
4. GitHub online repository for collaboration.
5. Tortoise SVN for version control.

Chapter 5

Implementation

5.1 Correct Sequences

Suppose the following are a run sequence and a frequent sequence. We need to find out if the frequent sequence is present in a run sequence. But how do we decide of all the combinations possible which ones to select?

5.1.1 Run Sequence

Run Sequences are provided as input in a JSON file. They denote the order of the method invocations of the acceptance test.

run sequence	1	4	5	4	36	84	1	3	4	5	8	17	99	8	32	9
counters	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

5.1.2 Frequent Sequence

Frequent Sequence is one of the many output sequences that BIDE provide in frequent.dat file. It is closed and maximal Suppose one of the frequent sequence is 1 4 5

5.1.3 Analysis

So let's see which of the few possible sequences are correct or useful for us.
As there can be repetetive functions for the same run sequence, we will uniquely identify the method with respect to the counter number.

✓ 0 1 2

This is a straightforward consecutive sequence and is valid.

✓ 0 3 9

This is also a valid sequence as 1 4 5 appear in order though they are not consecutive.

X 6 3 9

This will not be considered valid because though the counters suggest 1 4 5 but they are not in order and with respect to integration testing run sequences are ordered so in other words we donot have the state of functions in that order and thus we are not interested in these types of sequences as they do not occur.

5.2 Block Structure

The Fig.5.1 explains the basic blocks of the project.
They consist of various sub functions and/or helping functions that we will see in the detailed explanation.

5.2.1 Generate Settings File

Before starting the program one needs to set the values of Settings.java file.
This is a static class which holds various values which are user inputs and which remain constant throughout the program.
They are:

- basePath : The folder where all the SUT folders reside
- subjectAppName : The name of the SUT (to be appended to basePath)

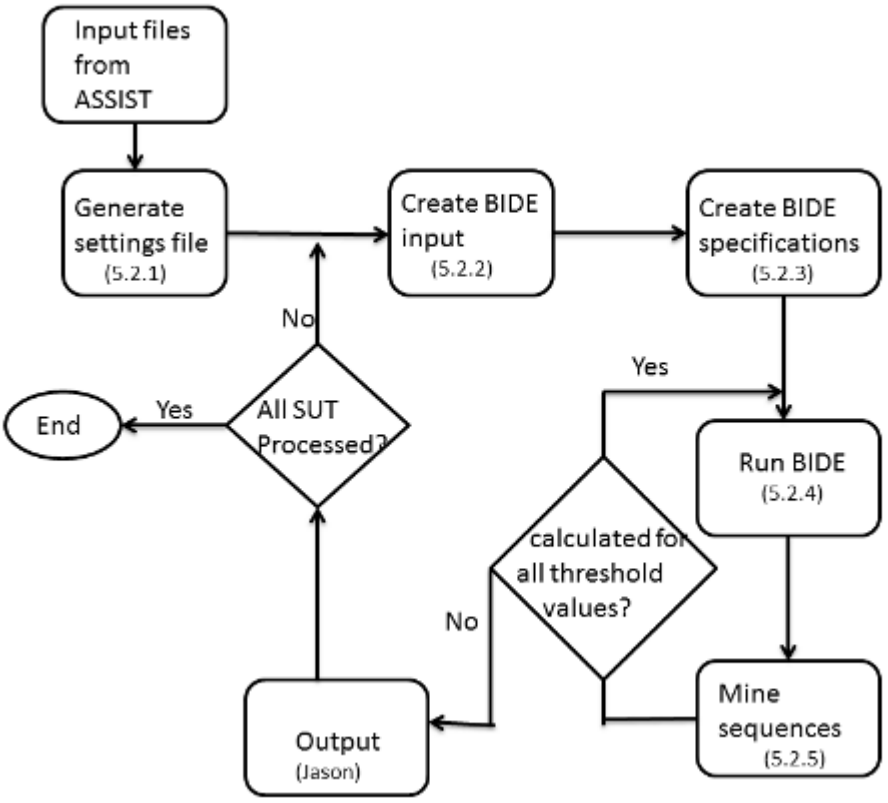

FIGURE 5.1: Project Implementation.

- thresholdValues : The array which stores all the threshold values for BIDE for which the SUT will be tested and results will be stored under the folder basePath/subjectAppName/thresholdValue.
- methodMapFile : File name format for the input JSON file which maps each method with its name and class name to a unique number
- sequenceMapFile : File name format for the Run Sequences in JSON file.

and similar BIDE related data ...

5.2.2 Create BIDE Input

This module takes care of converting the Run Sequences in input files from the JSON format to the textual-dataset (format required by the BIDE), explained in the Tools Section.

For each Run Sequence the data is extracted from JSON packets and stored in an BIDE input file.

5.2.3 Create BIDE Specification

While parsing the run sequences, ststistics required by the specification file format of BIDE are calculated. They are:

- Number of Unique Items(Total number of methods, obtained from methodMap)
- Number of Sequences(Total number of Run Sequences in input file)
- Maximum length of sequences(Keeping a track while parsing from JSON)
- Average length of sequence(Keeping a track while parsing)

5.2.4 Run BIDE

This module is called for each value of threshold mentioned in the Settings file.
An Operating System Process is borrowed and the BIDE executable is run on a Runtime Data Structure each time the module is called. There are two types of executables, one is with output on console and one is without output. The one with output can be used for logging purposes.

5.2.5 Mine Sequences

This module is the heart of the project. Most of the computative things happen in this module. There are various helper and subfunctions in this module. We will discuss them in detail in the following subsections.

Global Parameters for the module : {JSONObject, JSONArray, iterator, JSONParser}

Let us see a combined pseudocde for an overview:

5.2.5.1 getOccurence()

parameters : Frequent Sequence

return value : All Subsequences from all the Run Sequences

body : Iterates over all Run Sequences and calls a helper function getSubSequences()

5.2.5.2 getSubSequence()

parameters : Frequent Sequence, Run Sequence, Counters of Run Sequence

return value : All the Subsequences of a particular Run Sequence

body : Calls getIndividualSet()

```
do{
    add another item and create all possible combinations
    check if combination is a correct one
    reject the incorrect combination
}
while(all items of frequent set covered)
```

This solution is inspired from the classic computer NP-Hard problem of Longest Common Subsequence[22].

5.2.5.3 getIndividualSet()

parameters : Frequent Sequence, Run Sequence, Counters of Run Sequence

return value : Sets, for each item in the frequent set, consisting of all the occurences in run sequence

body :

```
for(each item in frequent set){
    for(each item in Run Sequence){
        if(items match){
            add counter in the return list
        }
    }
}
```

Chapter 6

Conclusion

6.1 Future Work

There are several shortcomings to the present version of the project. Following are the enhancements that can be worked upon:

- Newer heuristics similar to shortest distance can be devised to further help develop better integration tests.
- Automatic heuristic development can be thought of on the basis of interactions within the AUT.
- Threshold for a particular AUT varies on the basis of the spread of the Run Sequences. Ways can be devised to intelligently approximate the values to be tested.

6.2 Conclusion

In Conclusion MINUSA helps ASSIST in mining useful sequences and pruning less useful sequences from a very big corpus generated. Although the task is resource and time intensive, efficient heuristics can be applied and the aim can be achieved in an acceptable time frame. Results of an application dealing in nanoscience are shown in the AppendixA along with the code files.

Appendix A

Results and Code

A.1 Results

As seen in the Fig.A.1 method-sequence is the Frequent Sequence and sequence number denotes the Run Sequence and thus for each Frequent Sequence all occurences from all Run Sequence are shown. The list is exhaustive even for the shortest distance hueristic and thus only a snippet is shown.

All the results and code are available at the Online repository.


```
[{"method-sequence":[51,86,16,81,53,69],
    "occurences":[{"sequence-no":0,"counters":[34556,34557,34558,34559,345
    {"sequence-no":1,"counters":[30812,30813,30814,30815,30816,30817]},
    {"sequence-no":4,"counters":[27816,27817,27818,27819,27820,27821]},
    {"sequence-no":6,"counters":[23326,23327,23328,23329,23330,23331]},
    {"sequence-no":7,"counters":[21751,21752,21753,21754,21755,21756]},
    {"sequence-no":8,"counters":[2795,2796,2797,2800,2801,2802]},
    {"sequence-no":13,"counters":[12839,12840,12841,12842,12843,12844]},
    {"sequence-no":17,"counters":[4459,4460,4461,4462,4463,4464]},
    {"sequence-no":19,"counters":[10600,10601,10602,10603,10604,10605]},
    {"sequence-no":21,"counters":[28318,28319,28320,28321,28322,28323]},
    {"sequence-no":28,"counters":[6115,6116,6117,6118,6119,6120]},
    {"sequence-no":30,"counters":[17301,17302,17303,17304,17305,17306]},
    {"sequence-no":32,"counters":[15071,15072,15073,15074,15075,15080]},
    {"sequence-no":35,"counters":[2238,2239,2240,2241,2242,2243]},
    {"sequence-no":40,"counters":[28663,28664,28665,28666,28667,28668]},
    {"sequence-no":43,"counters":[8366,8367,8368,8369,8370,8371]},
    {"sequence-no":47,"counters":[32967,32968,32969,32970,32971,32972]},
    {"sequence-no":49,"counters":[1324,1325,1326,1327,1328,1329]},
    {"sequence-no":52,"counters":[24904,24905,24906,24907,24908,24909]},
    {"sequence-no":53,"counters":[13,14,15,16,17,18]},
```


FIGURE A.1: Results of one of the thresholds for Shortest SubSequence.

A.2 Settings.java

```
import org.apache.commons.io.FilenameUtils;

public class Settings {

    public static String basePath = "C:\\MyData\\";
    public static String subjectAppName = "nanoxml";
    public static int rank = 10;

    //nanoxml
    public static String bide_threshold_values[] = {"0.11", "0.12", "0.13", "0.14
    ", "0.15", "0.16", "0.17", "0.18", "0.19", "0.20" };


    //input files to generating bide input files
    public static String methodMapFile = FilenameUtils.concat(basePath,
    subjectAppName +"\\"+subjectAppName +  "-" + "method-map.json");
    public static String methodSequenceFile = FilenameUtils.concat(basePath,
    subjectAppName +"\\"+subjectAppName +  "-" +  "sequence.json");

    //BIDE related data
    public static String bideExecFolder = "C:\\Users\\Kush\\Documents\\CS598\\
    BIDE\\";
    public static String bideInputFileName = FilenameUtils.concat(basePath,
    subjectAppName+"\\"+subjectAppName + "-ip.txt");
    public static String bideSpecFileName = FilenameUtils.concat(basePath,
    subjectAppName +"\\"+subjectAppName+ "-ip.spec");
    public static String batchFileNameBase = FilenameUtils.concat(basePath,
    subjectAppName  + "\\");
    public static String batchFileNameEnd =   "\\"  + "cmd.bat";
    public static String bideOutputFileNameBase = FilenameUtils.concat(basePath,
    subjectAppName + "\\" );
    public static String bideOutputFileNameEnd =  "\\"+"frequent.dat";


    //BIDE output converted to methods

    public static String jsonOutputFileNameBase = FilenameUtils.concat(basePath,
    subjectAppName +"\\");
    public static String jsonOutputFileNameEnd = "\\"+subjectAppName+"-" + "OP.
    json";

}
```

Listing A.1: Settings.java

A.3 MyTest.java

```
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
import org.json.simple.parser.ContainerFactory;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.*;
import java.util.*;

/**
 * Created by Kush on 2/3/14.
 */

public class MyTest {
    Object obj;
    JSONArray array;
    Iterator iterator;
    JSONParser parser;
    ContainerFactory containerFactory;
    HashMap<String, Integer > methodHashMap;
    BufferedWriter bufferedWriter;
    int seqCount;
    int maxSeq;
    float seqTotal;
    int noOfmthds;

    public MyTest() {
        this.containerFactory = new ContainerFactory(){
            public List creatArrayContainer() {
                return new ArrayList();
            }

            public Map createObjectContainer() {
                return new LinkedHashMap();
            }

        };
        this.methodHashMap = new HashMap<String, Integer>();
        this.parser = new JSONParser();
        seqCount = 0;
        maxSeq = 0;
        seqTotal = 0;
        noOfmthds = 0;
    }
```

```
    public void generateOP(){
        createBIDEIPfromSeqMap();
        createBIDESpec();
        for(String threshold : Settings.bide_threshold_values){
            File file = new File(Settings.basePath + Settings.subjectAppName + "
    \\" + threshold);
            file.mkdir();
            runBIDE(threshold);
            formatBIDEOP(threshold);
        }

    }

    private void createBIDEIPfromSeqMap() {
        try {
            obj = JSONValue.parse(new FileReader(Settings.methodMapFile));
            array=(JSONArray)obj;
            noOfmthds = array.size();
            boolean flag = false;
            JSONArray seq = new JSONArray();
            bufferedWriter = new BufferedWriter(new FileWriter(Settings.
    bideInputFileName));
            obj = JSONValue.parse(new FileReader(Settings.methodSequenceFile));
            array=(JSONArray)obj;
            iterator = array.iterator();
            while (iterator.hasNext()){
                String line = iterator.next().toString();
                Map lineMap = (Map)parser.parse(line, containerFactory);
//                System.out.println(lineMap.get("method-id"));
                String mthdString = lineMap.get("method-id").toString();
                int tempMthdCount = 0;
                for(String str : mthdString.substring(1,mthdString.length()-1).
    split(", ")){
//                    System.out.println(str);
                    bufferedWriter.write(str + " ");
                    tempMthdCount = tempMthdCount + 1;
                }
                bufferedWriter.write("-1");
                bufferedWriter.newLine();
                if(tempMthdCount > this.maxSeq){
                    this.maxSeq = tempMthdCount;
                }
                this.seqTotal = this.seqTotal + tempMthdCount;
                this.seqCount = this.seqCount + 1;
            }
            bufferedWriter.write("-2");
            bufferedWriter.close();
        } catch (IOException e) {
```

```
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }

    private void createBIDESpec() {
        try {
            bufferedWriter = new BufferedWriter(new FileWriter(Settings.
    bideSpecFileName));
//            System.out.print(Settings.bideInputFileName);
//            System.out.println("haha");
            bufferedWriter.write(Settings.bideInputFileName.trim());
            bufferedWriter.newLine();
            bufferedWriter.write(String.valueOf(noOfmthds));
            bufferedWriter.newLine();
            bufferedWriter.write(String.valueOf(seqCount));
            bufferedWriter.newLine();
            bufferedWriter.write(String.valueOf(maxSeq));
            bufferedWriter.newLine();
            bufferedWriter.write(String.valueOf(Math.round(seqTotal / seqCount)))
    ;
            bufferedWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        MyTest test = new MyTest();
        test.generateOP();
    }

    private void formatBIDEOP(String threshold) {
        try {
            BufferedWriter bideOP = new BufferedWriter(new FileWriter(Settings.
    jsonOutputFileNameBase + threshold + Settings.jsonOutputFileNameEnd));
            bideOP.write("[");
            Scanner sc = new Scanner(new FileReader(Settings.
    bideOutputFileNameBase + threshold + Settings.bideOutputFileNameEnd));
            JSONArray mainArray = new JSONArray();
            int occCount = 0;
            while (sc.hasNextLine()){
                JSONObject  lineObject = new JSONObject();
                String bideOPLine = sc.nextLine();
//                System.out.println(bideOPLine);
                String [] seqArray = bideOPLine.split(":")[0].split(" ");
                JSONArray seqJSONArray = new JSONArray();
```

```
                JSONArray occurencesJSONArray;
                for(String each : seqArray){
                    seqJSONArray.add(Integer.parseInt(each));
                }
                lineObject.put("method-sequence", seqJSONArray);
                occurencesJSONArray = getOccurrences(seqArray);
                //getOccurrences(seqArray);
                lineObject.put("occurences", occurencesJSONArray);
//                System.out.println(Arrays.toString(seqArray) +":"+
    occurencesJSONArray.size());
                occCount = occCount + occurencesJSONArray.size();
//                System.out.println("Total:"+occCount);
//                System.out.println(lineObject.toJSONString());
//                Iterator iterator = occurencesJSONArray.iterator();
//                while(iterator.hasNext()){
//                    System.out.println(iterator.next());
//
//                }
                bideOP.append(lineObject.toJSONString() + ",");
                bideOP.newLine();
            }
            bideOP.append("]");
            bideOP.close();
            //bideOP.write(mainArray.toJSONString());
            //bideOP.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private JSONArray getOccurrences(String[] seqArray) {
        JSONArray mainArray = new JSONArray();
        try {
            Object obj= JSONValue.parse(new FileReader(Settings.
    methodSequenceFile));
            JSONArray array=(JSONArray)obj;
            Iterator iterator = array.iterator();
//              JSONObject occurencesObject = new JSONObject();
            while (iterator.hasNext()){
                String line = iterator.next().toString();
                Map lineMap = (Map)parser.parse(line, containerFactory);
//                System.out.println(line);
//                System.out.println(lineMap.get("counters"));
                ArrayList<String> counterList = (ArrayList)parser.parse(lineMap.
    get("counters").toString(),containerFactory);
```

```
                ArrayList<String> ctrArrayTemp = new ArrayList<String>(
    counterList);
                ArrayList<String> ctrArray = new ArrayList<String>();
                for(Object s: ctrArrayTemp){
                    ctrArray.add(s.toString());
                }
                ArrayList<String> methodList = (ArrayList<String>)parser.parse(
    lineMap.get("method-id").toString(),containerFactory);
                ArrayList<String> mthdArrayTemp = new ArrayList<String>(
    methodList);
                ArrayList<String> mthdArray = new ArrayList<String>();
                for(Object s: mthdArrayTemp){
                    mthdArray.add(s.toString());
                }
                //String [] mthdArray = methodList.toArray(new String[methodList.
    size()]);
//                ArrayList<ArrayList<String>> occurence = getOccurence(seqArray,
    mthdArray, ctrArray);
//                if(seqArray.length > mthdArray.size()){
//                    ArrayList<ArrayList<String>> occurence = getSubSequences(
    seqArray,mthdArray, ctrArray);
//                    ArrayList<ArrayList<String>> occurence =
    getShortestSubSequences(seqArray,mthdArray, ctrArray);
                ArrayList<ArrayList<String>> occurence =
    getShortestRankedSubSequences(seqArray, mthdArray, ctrArray);
    //                if(occurence.size() != 0){
    //                System.out.println(lineMap.get("sequence-id")+":"+occurence
    .size());
    //                 }
    //                System.out.println("I am here");
    //                JSONArray counterArray = new JSONArray();
                    for(ArrayList<String> lstr : occurence){
                        JSONArray counterArray = new JSONArray();
                        for(String str: lstr){
                            counterArray.add(Integer.parseInt(str));
                        }
                        JSONObject occurencesObject = new JSONObject();
    //                    occurencesObject.put("counters",counterArray.clone());
                        occurencesObject.put("counters",counterArray);
                        occurencesObject.put("sequence-no",lineMap.get("sequence-
    id"));
                        mainArray.add(occurencesObject);
    //                    mainArray.add(occurencesObject.clone());
    //                    occurencesObject.clear();
    //                    counterArray.clear();
                    }
                // }
            }
```

```
        }catch (ParseException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return mainArray;
    }

    private ArrayList<ArrayList<String>> getShortestRankedSubSequences(String[]
    seqArray, ArrayList<String> mthdArray, ArrayList<String> ctrArray) {
        ArrayList<ArrayList<String>> listOfOccurrences = getIndividualSet(
    seqArray, mthdArray, ctrArray);
        ArrayList<ArrayList<String>> result = new ArrayList<ArrayList<String>>();
        for(String each : listOfOccurrences.get(0)){
            result.add(new ArrayList<String>(Arrays.asList(each)));
        }
        for(int i =1; i < listOfOccurrences.size();i++){
            ArrayList<ArrayList<String>> temp = new ArrayList<ArrayList<String
    >>();
            for(String comparer : listOfOccurrences.get(i)){
                for(ArrayList<String> partSeq : result){
                    if(Integer.parseInt(partSeq.get(partSeq.size()-1)) < Integer.
    parseInt(comparer)){
                        ArrayList<String> tempArr = new ArrayList<String>(partSeq
    );
                        tempArr.add(comparer);
                        temp.add(tempArr);
                    }
                }
            }
            result = temp;
        }
        int difference = Integer.MAX_VALUE;
        int counter = 0;
        ArrayList<ArrayList<String>> newResult = new ArrayList<ArrayList<String
    >>();
        for(int index = 0;index < Settings.rank;index++){
            for(int i = 0;i<result.size();i++){
                if((Integer.parseInt(result.get(i).get(result.get(i).size()-1)) -
     Integer.parseInt(result.get(i).get(0))) < difference){
                    difference = (Integer.parseInt(result.get(i).get(result.get(i
    ).size()-1)) - Integer.parseInt(result.get(i).get(0)));
                    counter = i;
                }
            }
            if(result.size() != 0){
                newResult.add(result.get(counter));
                result.remove(counter);
```

```
                difference = Integer.MAX_VALUE;
                counter = 0;
            }
        }
        return newResult;
    }

    private ArrayList<ArrayList<String>> getShortestSubSequences(String[]
    seqArray, ArrayList<String> mthdArray, ArrayList<String> ctrArray) {
        ArrayList<ArrayList<String>> listOfOccurrences = getIndividualSet(
    seqArray, mthdArray, ctrArray);
        ArrayList<ArrayList<String>> result = new ArrayList<ArrayList<String>>();
        for(String each : listOfOccurrences.get(0)){
            result.add(new ArrayList<String>(Arrays.asList(each)));
        }
        for(int i =1; i < listOfOccurrences.size();i++){
            ArrayList<ArrayList<String>> temp = new ArrayList<ArrayList<String
    >>();
            for(String comparer : listOfOccurrences.get(i)){
                for(ArrayList<String> partSeq : result){
                    if(Integer.parseInt(partSeq.get(partSeq.size()-1)) < Integer.
    parseInt(comparer)){
                        ArrayList<String> tempArr = new ArrayList<String>(partSeq
    );
                        tempArr.add(comparer);
                        temp.add(tempArr);
                    }
                }
            }
            result = temp;
        }
        int difference = Integer.MAX_VALUE;
        int counter = 0;
        ArrayList<ArrayList<String>> newResult = new ArrayList<ArrayList<String
    >>();
        for(int i = 0;i<result.size();i++){
            if((Integer.parseInt(result.get(i).get(result.get(i).size()-1)) -
    Integer.parseInt(result.get(i).get(0))) < difference){
                difference = (Integer.parseInt(result.get(i).get(result.get(i).
    size()-1)) - Integer.parseInt(result.get(i).get(0)));
                counter = i;
            }
        }
        if(result.size() != 0){
            newResult.add(result.get(counter));
        }
        return newResult;
    }
```

```
public ArrayList<ArrayList<String>> getSubSequences(String[] seqArray,
ArrayList<String> mthdArray, ArrayList<String> ctrArray) {
    ArrayList<ArrayList<String>> listOfOccurrences = getIndividualSet(
seqArray, mthdArray, ctrArray);
    ArrayList<ArrayList<String>> result = new ArrayList<ArrayList<String>>();
    for(String each : listOfOccurrences.get(0)){
        result.add(new ArrayList<String>(Arrays.asList(each)));
    }
    for(int i =1; i < listOfOccurrences.size();i++){
        ArrayList<ArrayList<String>> temp = new ArrayList<ArrayList<String
>>();
        for(String comparer : listOfOccurrences.get(i)){
            for(ArrayList<String> partSeq : result){
                if(Integer.parseInt(partSeq.get(partSeq.size()-1)) < Integer.
parseInt(comparer)){
                    ArrayList<String> tempArr = new ArrayList<String>(partSeq
);
                    tempArr.add(comparer);
                    temp.add(tempArr);
                }
            }
        }
        result = temp;
    }

    return result;
}

public ArrayList<ArrayList<String>> getIndividualSet(String[] seqArray,
ArrayList<String> mthdList, ArrayList<String> ctrList) {
    ArrayList<ArrayList<String>> tempList = new ArrayList<ArrayList<String
>>();
    ArrayList<String> set = new ArrayList<String>();
    for(String seq : seqArray){
        int i=0;
        for (Iterator<String> iter = mthdList.iterator(); iter.hasNext(); i
++) {
            String obj = iter.next();
            String st =String.valueOf(obj);
            //                System.out.println(st + ":" + seq);

            if(seq.trim().equals(st.trim())){
                //                    System.out.println("str:"+st+"in:"+
mthdList.indexOf(st));
                set.add(ctrList.get(i));
```

```
                    //                    System.out.println(Arrays.toString(set.
    toArray()));
                }
            }

            //            System.out.println(Arrays.toString(set.toArray()));
            tempList.add(new ArrayList<String>(set));
            set.clear();
        }

        return tempList;
    }

    public ArrayList<ArrayList<String>> getCartesianProduct(ArrayList<ArrayList<
    String>> tempList) {
        int n = tempList.size();
        int solutions = 1;

        for(ArrayList<String> vector : tempList) {
            solutions *= vector.size();
        }

        //String[][] allCombinations = new String[solutions + 1][];
        ArrayList<ArrayList<String>> allCombinations = new ArrayList<ArrayList<
    String>>();
        //allCombinations[0] = dataStructure.keySet().toArray(new String[n]);
        boolean breakFlag = false;
        int count = 0;
        for(int i = 0; i < solutions; i++) {
            breakFlag = false;
//            count = count + 1;
            int prev = -1;
            ArrayList<String> combination = new ArrayList<String>(n);
            int j = 1;
            for(ArrayList<String> vec : tempList) {
                int curr = Integer.parseInt(vec.get((i / j) % vec.size()));
                if(curr <= prev){
                    breakFlag = true;
                    break;
                }
                combination.add(vec.get((i / j) % vec.size()));
                prev = Integer.parseInt(vec.get((i / j) % vec.size()));
                j *= vec.size();

            }
            if(breakFlag){
                continue;
            }
```

```
            allCombinations.add(new ArrayList<String>(combination));
        }
//        System.out.println(count);
        return allCombinations;
    }

    public boolean isIncreasing(String s) {
        s = s.substring(1,s.length()-1);
        String[] sarr = s.split(",");
//        System.out.println(Arrays.toString(sarr));
        String previous = "";
        int prev = -1;
        for (final String current: sarr) {
//            System.out.println("current:" + current + "previous:" + previous +
    ":"+current.trim().compareTo(previous));
            if (Integer.parseInt(current.trim()) <= prev){
                return false;
            }
            prev = Integer.parseInt(current.trim());
        }

        return true;
    }

    public List<HashSet<String>> getTempList(String[] seqArray, ArrayList<String>
     mthdList, ArrayList<String> ctrList) {
        List<HashSet<String>> tempList = new ArrayList<HashSet<String>>();
        for(String seq : seqArray){
            HashSet<String> set = new HashSet<String>();
            int i=0;
            for (Iterator<String> iter = mthdList.iterator(); iter.hasNext(); i
    ++) {
                Object obj = iter.next();
                String st =String.valueOf(obj);
                //                System.out.println(st + ":" + seq);

                if(seq.trim().equals(st.trim())){
                    //                    System.out.println("str:"+st+"in:"+
    mthdList.indexOf(st));
                    set.add(ctrList.get(i));

                    //                    System.out.println(Arrays.toString(set.
    toArray()));
                }
            }

            //            System.out.println(Arrays.toString(set.toArray()));
            tempList.add((HashSet)set.clone());
```

```
            //set.clear();
        }

        return tempList;
    }

    private void runBIDE(String threshold) {
        String line;
        Process pr = null;
        try {
//            comment the following 3 lines if direct command from another file(
    mac)
            BufferedWriter bw = new BufferedWriter(new FileWriter(Settings.
    batchFileNameBase + threshold + Settings.batchFileNameEnd));
            bw.write("cd " + Settings.basePath + Settings.subjectAppName + "\\" +
     threshold);
            bw.newLine();
            bw.write(Settings.bideExecFolder + "BIDEwithOUTPUT.exe " + Settings.
    bideSpecFileName + " " + threshold);
            bw.close();
//            change file name here for direct execution
            pr = Runtime.getRuntime().exec(Settings.batchFileNameBase + threshold
     + Settings.batchFileNameEnd);

            BufferedReader Resultset = new BufferedReader(
                    new InputStreamReader (
                            pr.getInputStream()));

            while((line=Resultset.readLine())!=null)
            {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

Listing A.2: MyTest.java

Bibliography

[1] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, pages 79–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2065-0. URL http://dl.acm.org/citation.cfm?id=977401.978142.

[2] Luciano Baresi and Michal Young. Test oracles. Technical Report CIS-TR-01-02, University of Oregon, Dept. of Computer and Information Science, Eugene, Oregon, U.S.A., August 2001. http://www.cs.uoregon.edu/~michal/pubs/oracles.html.

[3] Dennis Peters and David L. Parnas. Generating a test oracle from program documentation: work in progress. In *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pages 58–65, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-683-2. doi: http://doi.acm.org/10.1145/186258.186508.

[4] D. J. Richardson, S. Leif-Aha, and T. O. O'Malley. Specification-based Test Oracles for Reactive Systems. In *Proceedings of the 14th ICSE*, pages 105–118, May 1992.

[5] Debra J. Richardson. Taos: Testing with analysis and oracle support. In *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT ISSTA*, pages 138–153, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-683-2. doi: http://doi.acm.org/10.1145/186258.187158.

[6] Laura K. Dillon and Qing Yu. Oracles for checking temporal properties of concurrent systems. In *Proceedings of the ACM SIGSOFT '94*, pages 140–153, December 1994.

[7] Kevin M. Conroy, Mark Grechanik, Matthew Hellige, Edy S. Liongosari, and Qing Xie. Automatic test generation from gui applications for testing web services. In *ICSM*, pages 345–354, 2007.

[8] Brett Daniel, Vilas Jagannath, Danny Dig, and Darko Marinov. Reassert: Suggesting repairs for broken unit tests. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 433–444, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3891-4. doi: 10.1109/ASE.2009.17. URL http://dx.doi.org/10.1109/ASE.2009.17.

[9] Robert S. Boyer, Bernard Elspas, and Karl N. Levitt. Select - a formal system for testing and debugging programs by symbolic execution. In *Proceedings of the international conference on Reliable software*, pages 234–245, New York, NY, USA, 1975. ACM. doi: 10.1145/800027.808445. URL http://doi.acm.org/10.1145/800027.808445.

[10] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7): 385–394, 1976.

[11] Lori A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Trans. Software Eng.*, 2(3):215–222, 1976.

[12] Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. DySy: Dynamic symbolic execution for invariant inference. In *Proc. 30th ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 281–290. ACM, May 2008.

[13] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: Directed automated random testing. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 213–223. ACM, June 2005.

[14] Koushik Sen and Gul Agha. Cute and jCute: Concolic unit testing and explicit path model-checking tools. In *Proc. International Conference on Computer Aided Verification (CAV)*, pages 419–423. Springer, August 2006.

[15] Lori A. Clarke and Debra J. Richardson, editors. *Symbolic evaluation methods for program analysis.*, 1981. Prentice-Hall.

[16] James C. King. A program verifier. In *IFIP Congress (1)*, pages 234–249, 1971.

[17] .

[18] https://code.google.com/p/json-simple/, .

[19] http://maven.apache.org/what-is-maven.html.

[20] https://maven.apache.org/guides/introduction/introduction-to-the-pom.html.

[21] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1): 55–86, August 2007. ISSN 1384-5810. doi: 10.1007/s10618-006-0059-1. URL http://dx.doi.org/10.1007/s10618-006-0059-1.

[22] http://en.wikipedia.org/wiki/longest_common_subsequence_problem.