Graph Theoretical Algorithms For Structural Comparison Of Java Source And Byte Code

---

Submitted By

Artem Garishin

FB2: Faculty of Computer Science and Engineering

This thesis presented for the degree of
Master of Science
in the

High Integrity Systems

Research Supervisor: Prof. Dr. Alekseev Sergej

Co-Supervisor: Prof. Dr. Matthias Wagner

August 2014

Legal Declaration

I declare that this thesis document is completely my own work and all used references have been clearly cited. I have not submitted this assignment in the context of an examination to any other examination board or person.

Signature:

Location, Date:

Abstract

Java code is compare we need for ...

Why we need it

This paper also explains the existing ...

Acknowledgments

I would like to take this time to thank Frankfurt University of Applied Sciences for all of the resources which they provided me in order to pursuing my master study in computer science and make this thesis possible.

I would like to express my sincere gratitude to Prof. Dr. Sergej Alekseev and Prof. Dr. Matthias Wagner for their patient guidance, encouragement and advice which they provided me throughout this thesis work.

Contents

1 Introduction **1**

2 Description of problem **2**

3 Graphs comparisons algorithms **5**

3.1 An algorithm for Top-down maximum common sub-tree isomorphism 5

3.2 An algorithm for Bottom-Up maximum common sub-tree isomorphism 5

4 Code compare experiments **6**

4.1 Introduction to experiments 6

4.2 Experiments on Java source code 8

5 Graph transformation algorithms **9**

5.1 Introduction to the graph transformation 9

5.2 Techniques to build a tree from code 9

5.3 Convert graph to tree 9

5.4 Possible ideas??? 9

6 Conclusion **10**

Bibliography **11**

List of Figures

List of Tables

Abbreviations

IEDAE	**I**nteractive **E**xploratory **D**ata **A**nalysis **E**nvironment
SQL	**S**tructural **Q**uery **L**anguage
API	**A**pplication **P**rogramming **I**nterface
CI	**C**ontinuous **I**ntegration
JDBC	**J**ava **D**ata **B**ase **C**onnectivity
MVC	**M**odel **V**iew **C**ontroller
HTML	**H**yper **T**ext **M**arkup **L**anguage
XML	**E**xtensible **M**arkup **L**anguage
JAXB	**J**ava **A**rchitecture for **X**ML **B**inding
UML	**U**nified **M**odeling **L**anguage
URL	**U**niform **R**esource **L**ocator
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
SCM	**S**ource **C**ode **M**anagement
CVS	**C**oncurrent **V**ersion **S**ystem

Chapter. 1

Introduction

Modern methods to compare of programming pieces of code are used to analyze code's changing, to explore development process and so on. Basically in current tools or plug-ins only text compare methods are used, that is not full sufficient to define code compare. Sometimes another techniques can be very helpful for such purposes. One of them is a structural code compare, based on building a trees, and methods to compare any similar or same structures.

TODO START: You can't write a good introduction until you know what the body of the paper says. Consider writing the introductory section(s) after you have completed the rest of the paper, rather than before. Be sure to include a hook at the beginning of the introduction. This is a statement of something sufficiently interesting to motivate your reader to read the rest of the paper, it is an important/interesting scientific problem that your paper either solves or addresses. You should draw the reader in and make them want to read the rest of the paper.

Tips: A statement of the goal of the paper: why the study was undertaken, or why the paper was written. Do not repeat the abstract.

Chapter. 2

Description of problem

In this chapter an issue of the work is being explained. As usual a compare of two codes we can consider them as classes, functions or methods. Thereby a compare can be counted as examinations of two pieces of code, in the best case a methods or functions. They can have a similar implementation or alike syntax, however these two pieces of code are different.

There are many purposes to compare a code, to find out a similarity or determine a difference between them. One of the option is to search for plagiarism in case a code can be taken from external source and a variables have been changed. In addition to general search can be improved to look out a similar code in big projects.

If two graphs are being compared with each other, then it's NP - complete problem and takes much times and efforts to be done. Therefore this problem must to be reduced. Luckily, a tree comparison is able to executed in polynomial time and moreover there are some algorithms to compare them.

TODO: write what is NP-complete, write a problems with time execution

TODO: create a pieces of code and compare them

Thus there are no deterministic algorithms to compare them because of loops in graphs. In this case the input code can be transformed into graph firstly, after the graph creation, it must be converted into tree, using simple techniques removing back edges. The back edges are edges point out the same node.

After described code transformation a three questions can be asked: 1. how to transform code into tree optimally 2. how to compare these trees 3. how to reference code pieces and nodes(how to put the code difference)

Regards to the first question, the concept of idea is described in section "Graph Transformation". The second question comprehends existing algorithm and their combination and improvements in chapter "Existing algorithms". The last issue is about how to lead back the result of the code.

TODO: look in internet what exists already(code compare)

Possible result of this thesis is development of concept to find out code difference using graph theory, in the best case a tool in Project Dr. Garbage [1] . can be implemented that highlights similarity/difference of input code snippet and represented respective graph.

To get started searching optimal way to find similarity/difference two AST trees, compare them, find a logic how to link nodes with the code. Small example demonstrating, what kind of result gives text compare(just string compare)and AST:

TODO: highlight this code

public void ast1() if(i ¿ 1) i++;

public void ast2() if(i ¿ 1) i++;

In this example, two pieces of code there one enter symbol after line (i ¿ 1), therefore the codes look different. Using simple text compare approach, only these gap will be found, however this difference does not play any role regards business-logic. In Abstract Syntax View, these two graphs will be same, and no discrepancy will have been discovered.

TODO: create an example of AST and text compare

Since this article includes comparison not only of source code, from where an Abstract Syntax Tree can be easily build, but also Java byte code, where there is no syntax. Basically to have a look at byte code example, there are no bounds to hold a functions or methods. Based on this, control flow graph can be derived from byte code, that represents a graph, but not a spanning tree. Every node has a reference to byte code address.

TODO: is any idea how to optimize spanning tree(node has a byte code address, topologically sort)

TODO: find java bytecode example and his tree

IDEA: text difference highlight and in parallel nodes in Tree marks and Matching (TODO: make some experiments, if it's ok, better, worse or same compare)

To investigate code comparison, two algorithms of structural compare are required, in fact Top down maximum common subtree and Bottom Up maximum common subtree algorithms. To make a contribution into development of structured code compare, the following tasks should be explored:

1. The existing algorithms must be investigated (The text-compare method is not sufficient to find a similarity in code)
2. The algorithms for the structured compare(Abstract syntax trees, Control flow graphs) must be explored
3. New methods and algorithms find a place to tried out. A prototypes of combination text-compare and structure-compare can be implemented.
4. Experimental results of compare must be derived.

Chapter. 3

Graphs comparisons algorithms

3.1 An algorithm for Top-down maximum common subtree isomorphism

TODO: explain briefly what this technique, examples from Valiente

3.2 An algorithm for Bottom-Up maximum common subtree isomorphism

TODO: explain briefly what this technique, examples from Valiente

Chapter. 4

Code compare experiments

4.1 Introduction to experiments

For better understanding of possible "code compare" concept, possible ideas of implementation and following development an amount of experiments are required. In order to build a proper tool or at least a concept, in Eclipse plugins at Dr. Garbage Community® some hand experiments in code should be fulfilled.

All these test cases are performed in Eclipse IDE [4] and divided into blocks. These steps can be approached by following:

1. Research on Java source code using existing methods to compare:
   (a) Normal text compare
   (b) Spanning trees transformed from control flow graphs
2. Research on Java source code using existing methods to compare:
   (a) Normal text compare
   (b) Abstract syntax trees
3. Research on Java byte code using existing methods to compare:

(a) Normal text compare

(b) Control flow graphs

All test set are investigated under Java methods and functions. Playing around with the patterns of code changing variables, names, sequences of commands, adding loops or conditions and apply simple "text to text" compare. This "text compare is already implemented in Eclipse IDE, so-called command "compare with each other by member". This type of comparison provides a pop-up window, where two pieces of code are compared, line by line.

In parallel a control flow graphs or source graphs from the functions are being created and compared using implemented algorithms Top-Down and Bottom-Up(following called: TD& BU). The further task is to figure out the difference/similarity from graphical visual comparison. Consequently these both results must be matched and recorded for succeeding research.

The derived results from can be as follows:

1. Text compare and TD& BU have same difference
2. Text compare and TD& BU give similar difference
3. Text compare and TD& BU five full difference

Hence, as it was declared in section description of problem, based on these results can be decided what kind of tool in Dr. Garbage Eclipse plug-ins can be built.In case similar or full difference results, a combination of both methods can be used for optimal comparison.

Small example can be demonstrated: there are two functions that look very similar but nevertheless they have different number of string and different functionality. The abstract results can be following:

1. Text compare shows that strings 1 and 5 are different
2. Graph compare shows that string 7 is different

Conclusion: a combination of two methods can explicit that strings 1,5 and 7 are distinguished and much more distinction has been found. Thus it provides an optimal way of investigation.

4.2 Experiments on Java source code

In this section a sequence of actions and following statistic are represented. The steps are carried through sequence of action:

- Write two similar functions in Eclipse IDE
- Apply for them text-to-text comparison
- Declare the statistic, respectively how many lines are different
- Create a source-code graph for both
- Apply TD& BU algorithms to get structural difference
- Declare the statistic, respectively how many nodes are different

Chapter. 5

Graph transformation algorithms

5.1 Introduction to the graph transformation

TODO: explain how graph is generated, which libraries are used, plugins

5.2 Techniques to build a tree from code

TODO: what is AST, examples;

TODO: what is BasicBlock, examples;

5.3 Convert graph to tree

TODO: decribe here how to remove edges in order to get spanning tree

5.4 Possible ideas???

Chapter. 6

Conclusion

¡temp¿ based on experimental results and own opinion, write here what results were derived From time to time write here combined conclusions or improvements ¡/temp¿

Bibliography

[1] Sergej Alekseev, Peter Palaga and Sebastian Reschke, The Dr. Garbage Tools Project, 2013, http://www.drgarbage.com

[2] Sergej Alekseev, Graph theoretical algorithms for control flow graph comparison, 2013.

[3] Gabriel Valiente, Algorithms on Trees and Graphs, (Berlin: Springer-Verlag, 2002).

[4] Eclipse Integrated Development Environment, https://www.eclipse.org/

[5] Author, NAME, (Berlin: Springer-Verlag, 2002).