EECS 493 Research Proposal

Andrew Mason
ajm188@case.edu

Jon Pfeil
jwp69@case.edu

September 29, 2014

Contents

1 Related Work

1.1 gSpan

gSpan is an algorithm for frequent graph-based pattern mining. It explores depth-first search (DFS) in frequent subgraph mining using a DFS lexicographical ordering and minimum DFS codes, which form a canonical labeling system to support the DFS search [3]. The algorithm exploits the fact that, given a set of DFS codes for a graph, two graphs are isomorphic if and only if they have the same minimum DFS code.

1.2 GMFMM and IHMM

GMFMM and IHMM are two algorithms proposed by Chang and Podgurski for discovering both intraprocedural and interprocedural program dependencies. The algorithms model rules as graph minors of augmented dependence graphs, with edes to indicate shared data dependencies [1].

The first, GMFMM, is a greedy algorithm for rule discovery by mining maximal frequent minors from interprocedural dependence spheres [1]. The algorithm operates in two phases - the first finds nodes in frequent minors, and the second makes those frequent minors maximal by adding additional dependencies to the graph.

IHMM is the incremental heuristic minor matching algorithm, which discovers violations of rules, even if those instances are interprocedural. Given a user-specified key node n (usually a call-site node), IHMM searches n's base function and, if necessary, any ancestors of the base function to identify rule violations involving n [1]. IHMM also uses dependence spheres for finding correct instances and violations of programming rules. If IHMM cannot find a correct instance of the rule at the base function (by looking for an isomorphic minor), it looks to the ancestor functions, where it decides that a dependence sphere S contains a violation if the number of super-spheres containing no rule instance is greater than some user-specified threshold [1].

These algorithms are limited in that they will not detect rules whose elements are not connected by a data, control, or shared data dependency. The algorithms are also limited to looking within a given size bound for the dependence sphere [1].

2 Proposed Work

The goal of this project is to push forward the state of specification mining research. This will be done by applying frequent subgraph mining algorithms to Program Dependency Graphs that are generated via JPDG run on java source code. There are two measures of success for this goal:

1. Improve upon the speed of FSM PDG's with a baseline given by gSpan
2. Increase the number of non-trivial specifications mined over baseline specification mining systems.

To accomplish these goals, the specific work items must be done:

1. Complete literature surveys of specification mining and frequent subgraph mining.
2. Get a working implementation of JPDG to generate PDGs from Java source code.
3. Get a working implementation of the gSpan algorithm and apply it out-of-box to PDGs generated with JPDG.
4. Implement baseline specification mining systems.
5. Setup a testing framework to compare the speed of our mining algorithms and the number of non-trivial specifications mined.
6. Experiment with novel intermediate representations that will allow for specification mining across various levels of abstraction.

3 Methodology and Evaluation

The following is an outline of the methodology we will use to evaluate our specification miner:

1. Obtain a corpus of programs that already have their non-trivial specifications identified (this can be done by hand if we cannot obtain rights to a previously used data set)

2. Run gspan on the PDG generated by JPDG for each example program. Record the runtime as our speed baseline

3. Run multiple other specification miners on the program corpus. Record the number of non-trivial specifications mined by each miner on each example. These results will serve as our mining baselines

4. For each experimental specification miner we want to evaluate:

   (a) Run on each example in the program corpus

   (b) Record the runtime

   (c) Record the number of non-trivial specifications mined

4 Tools and Data Sources

4.1 Soot

Soot is a Java optimization framework, intended to be used as either a standalone tool to inspect class files, or to develop optimizations or transformations on Java byte code. Soot provides several different intermediate representations of Java byte code: [2]

- **Baf**: a streamlined representation of bytecode which is simple to manipulate.

- **Jimple**: a typed 3-address intermediate representation suitable for optimization. This is the primary representation used by Soot.

- **Shimple**: an SSA variation of Jimple.

- **Grimp**: an aggregated version of Jimple suitable for decompilation and code inspection.

- **Dava**: an abstract syntax tree-based representation. Produced via decompilation of the Jimple representation.

Intraprocedurally, the Soot framework is often used for its support for implementing intraprocedural data-flow analyses. Soot also provides call graph information for interprocedural analysis as part of its output. [2]

4.2 Heros

Heros is a an interprocedural, finite, distributive subset (IFDS) problem solving framework, that can be plugged into Java-based program analysis frameworks, like Soot.

4.3 Jasmin

Jasmin is a back end for the Soot framework. It is used to compile Soot from source.

4.4 Smali

Smali is an assembler and disassembler for the JVM. Additonally, it fully supports the full functionality of the dex format (Android executable files).

4.5 Parsemis

ParSeMiS is the parallel and sequential mining suite from the Friedrich-Alexander Universität Erlangen-Nüremburg. It utilizes parallel or specialized algorithms or heuristics to search for frequent, interesting substructures in graph datasets.

4.6 JPDG

JPDG is a tool developed by Tim Henderson which utilizes the Soot framework to generate procedure dependence graphs (PDGs) for Java programs. The PDGs can then be mined for frequent subgraphs to infer programming rules from source code.

4.7 Google Code

We will be using a collection of thousands of Java programs collected from Google Code by Tim Henderson for evaluating our research.

References

[1] Chang, R., and Podgurski, A. Discovering programming rules and violations by mining interprocedural dependences. *Journal of Software Maintenance 24*, 1 (2012), 51–66.

[2] Lam, P., Bodden, E., Lhoták, O., and Hendren, L. The Soot framework for Java program analysis: a retrospective. In *Cetus Users and Compiler Infrastructure Workshop* (Galveston Island, TX, October 2011).

[3] Yan, X., and Han, J. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining* (Washington, DC, USA, 2002), ICDM '02, IEEE Computer Society, pp. 721–.