Assignment 3: Genetic Improvement of Source Code with GIN

Formative, Weight(10%), Learning objectives (1,2,3),
Abstraction (4), Design (4), Communication (4), Data (5), Programming (5)

Due date: 11:59pm, 26 October 2018, Weight: 10.0 % of the course

1 Overview

Assignments should be done in groups consisting of four (4) students. If you have problems finding a group partner use the forum to search for group partners or contact the lecturer.

Quick Summary

Genetic improvement (GI) applies evolutionary algorithms to modify programs to improve some aspect of their behaviour. Program behaviours that we might use GI to improve can include: speed, correctness, accuracy, memory footprint and energy use.

The input to a GI process are:

1. The program P to be improved.
2. A set of test cases T which P can be run against.
3. A program M that measures the performance of P(T) (P ran against T).

The program M measures the performance of P in the way that we want the program improved. For example, if we want to improve program speed of P then M will simply time P(T) (whilst ensuring that P still passes all of its test cases). As another example, if we just wanted to improve the correctness of P then we would write M to count the number of test cases passed.

The output of the GI process is just a patch consisting of a sequence of edits to change the program. When the GI process finishes it applies the patch and presents us with the improved program for us to check and deploy.

For this assignment you will conduct a series of experiments in GI using a new GI framework called GIN ("Genetic Improvement in No-time"). GIN is small, simple, and easy to change - and it works on Java programs. At the moment GIN is set up to improve just one thing: program speed! However, it can also be used to fix simple errors in code and we'll use it for that as well.

For this assignment you will have to hand in a report briefly describing a series of four sets of experiments that you will do. In your experiments you will have to write some code – you will include a listing of the code you wrote at the end of your report.

Details of each section of the report follow but first we describe how to set up GIN.

Getting GIN

GIN is available online at: https://github.com/gintool/gin and the README.md file tells you what you need to do to run the project. However, to get things started faster – and because it is currently under development – you should download the GIN implementation included in the zip file attached to this assignment.[1] The zip file is simply the GIN distribution with the source code pre-compiled and some of the examples tweaked. To run GIN you will need to use a machine that has Java 1.8, JavaParser and JUnit. To modify GIN (which you will need to do in Exercise 3) you will also need Gradle. These all tend to work most smoothly on linux distributions and we have tested them on a linux virtual machine. If you are using the University machines then you can download the zip file into your home directory. We have collected some technical help in the course's Announcements and Discussions – please have a look there.

Note that we recommend that you use Linux, MacOS or a virtual machine with either operating system for this homework.

2 Assignment

Exercise 1 *Team work: who has done what? (zero points)*

We'd like each team member to write one paragraph about what he or she has contributed to this assignment. We will not mark this, and it will not have any effect on the marking of the other exercises. You might now ask "why do this then?" — well, through this no-stakes approach, we'd like to encourage self-regulation within the group and cooperative learning. You can't lose, you can only win.

Exercise 2 *Running GIN and measuring its performance (20 points)*

Download the GIN zip file and change to the gin-master/ directory. Type:

```
java -jar build/gin.jar examples/locoGP/SortBubbleDouble.java
```

this will call a program called LocalSearch.java which uses evolutionary search to build a set of patches to fix the SortBubbleDouble.java program. While searching GIN will run the test programs in SortBubbleDoubleTest.java to ensure that the patches don't break the program. The output of GIN is a list of patches that it tries. You can see that some of the patches break the program but some cause it to run faster. You might notice that the patches start with words like MOVE, COPY and DELETE. This is because GIN

[1] see https://myuni-canvas.adelaide.edu.au/files/3275065/download?download_frd=1

works by moving, copying and deleting existing lines of code in the program. It doesn't make up any new code.

When GIN has finished you should see a file called SortBubbleDouble.java.optimised in the examples/locoGP directory. Have a look at this file. Is this code better than the original program? If so, how?

For this part of the assignment you are to track and graph the performance of GIN on the following four programs in the locoGP directory:

- SortBubbleDouble.java
- SortBubbleLoops.java
- SortInsertion.java
- SortCocktail.java

The easiest way to collect the initial and running best times for the modified programs is to run with a "grep" filter afterwards, for example:

```
java -jar build/gin.jar examples/locoGP/SortInsertion.java | grep -E ".*(best|Initial)"
```

Run these experiments at least 15 times (ideally more often) for each of the four programs above. Collect and graph the data for the best times. After each run quickly check the optimised program and make a quick description of the change made (and whether it is one that an experienced programmer would make).

Present the setup and the results of your experiments, including the graphs in your report. This section must be no more than two pages. In this section make sure you include the aim of your experiment, the methodology of your experiment and the results and your interpretation of the results.

Exercise 3 *Analysing Patches (20 points)*

Choose one of the benchmarks you ran from Exercise 1 which regularly produced patches more than one edit long (Note: patches consisting of one edit only are not very interesting for this section). Run the benchmark multiple times. Record the edits in the final patch each time. Record which edits appear most often and the positions of those edits in the patch. Provide a visualisation of this data in your report.

Look at the source code of the chosen benchmark program and assess the effect of the edits that your multiple runs have found. Do they have the same effect as the edits in other patches? From your observations is it possible to deduce a shortened version of a patch which achieves the same result as the derived patches? If there is a shorter version (or versions) of a patch describe this patch and what it does in your report.

As with the first section, describe your aims, your methodology, your results and your analysis in your report. Again, this section should be no more than two pages long.

Exercise 4 *Minimising Patches (40 points)*

This exercise is more challenging! For this part of the assignment you will have to modify the source code of GIN to post-process the patches found so that they contain the minimum number of edits necessary to produce the same final optimised program. To do this you need to write code that takes the final patch and uses search to find which edits in that patch have no effect on the final optimised program. The choice of search technique is up to you but it has to be effective and it should be efficient. Note that, because there are multiple patches that will produce the same final program your modifications won't always produce the same edits but the edits that your search retains must produce the same result as the original patch.

For this part of the assignment you will need to edit the source code for GIN. As a starting point read the search() method in the file:

gin-master/src/main/java/gin/LocalSearch.java

The patch minimisation process should be called after the search has finished and the best patch is found.

For this section of your report you must briefly describe what your changes aim to do, describe your methodology, in this case, the algorithm you are implementing and the data structures it is working over. You should also briefly present your results. In the appendix of the report you must include a listing of the source code changes that you have made to GIN.

Exercise 5 *Your Own Case-Study (20 points)*

Here, you must create your own case study program to run GIN against. To do this you will need to provide a benchmark program that is not optimally efficient and run GIN against it in order to improve its efficiency. The program that you are modifying can be your own or it can be published and publicly available code (you must provide adequate references in this latter case). You are allowed to modify the code to introduce a source of inefficiency or - even better - you can use an example where the inefficiency was pre-existing.

For this section of the report you should introduce the program (include the program source in the appendix of the report), the source of inefficiency and then present the results of running GIN against this program. If your benchmark program has multiple sources of inefficiency or there are multiple ways to optimise the program then your analysis can be staged so that GIN first runs against simpler variants of the program before running against more challenging versions.

3 General procedure for handing in the assignment

Work should be handed in using the course website. The submission should include:

- pdf file of your report
- all source files (if you created any) including the modified GIN implementation
- a file name README.txt that contains instructions to run the code (if any), the names, student numbers, and email addresses of the group members
- for each group, there should be only 1 submission

Acknowledgment

Many thanks go to Brad Alexander for contributing this assignment.