Programming Java for Longer Battery Life

Anna Frances Rasburn

*October 2018*

**Interim Project Report**

Computing Science and Mathematics

University of Stirling

# Introduction

As the dependence on mobile phones and technology increases, we need to consider the environmental impact of this new reliance. Users now have a computer in their pocket and wish for their mobile phones to last longer, making them more convenient. A significant drain on battery are the applications that have a higher demand on the device’s CPU. Therefore this project’s aim is to research and investigate the different ways we can program Java applications (particularly regarding Android mobile applications) to reduce CPU power and therefore use less battery life.

This project is going to look at the different techniques that can be used to optimise java in a regards to energy and execution time. Specially the project are focusing on using genetic improvement and a change in operators on its effects on source code and it’s efficiency.

## Background and Context

This research project builds on research done previously by Dr Alexander Brownlee et al in their paper called ‘Object-oriented genetic improvement for improved Energy consumption in google guava’. The project also uses research done previously in Bruce’s et al’s paper named ‘Reducing Energy Consumption using Genetic Improvement’.

### Optimisation regarding Java

The definition of optimisation is ‘the action of making the best or most effective use of a situation or resource’. When regarding Java, it is the ways we can optimise it (or in this project at least) is looking how we can change Java code and Projects so we can improve their execution time so it is faster and also look at how we can make applications that use less energy.

In 1.1.3, we look at Genetic Algorithms which uses the notion of fitness. For any optimisation problem, the fitness function needs to be defined beforehand. Using different fitness functions can lead to different ways we can optimise Java so by changing the fitness to look at execution time or energy, Java can be optimised in different ways.

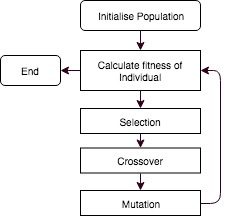
### Genetic Improvement regarding Search

Genetic Improvement is a way of improving software through using optimisation and machine learning techniques. These techniques include Evolutionary Algorithms and Genetic Algorithms. These techniques are part of Search Based Software Engineering which is a search technique using metaheuristics and genetic programming for optimisation problems.

### Genetic Algorithms

Genetic Algorithms are algorithms based on the biological notation of evolution. They are the most common version of Evolutionary algorithms. Genetic algorithms act based on the idea of natural selection where the fittest traits of the parent is passed on to the children. Running an algorithm multiple times improves the algorithm as the fittest traits are kept and traits that are not desired are disregarded.

In most Genetic Algorithms, there tends to be at least five steps that they follow, these are: initializing population, fitness, selection crossover and mutation. These steps are shown in Figure 1.

****

**Figure 1 Flow Diagram of Genetic Algorithm**

Individual is what is trying to find a solution to the problem. Individuals are comprised of chromosomes. A chromosome is thought of as a string of bits, each individual bit is a gene. In the mutation stage, the genes are flipped from one to zero or vice versa to get a new mutation of a chromosome to pass onto future populations. Genes are also thought of as parameters.

A Population is a set of individuals. As individuals get fitter, the overall population gets fitter and more appropriate to solve the problem what you have. In the GIN code, an edit is an example of a population as it is comprised of lots of individual code changes which are the individuals. A patch is a set of populations as it is comprised of a collection of edits.

## [Scope](http://www.cs.stir.ac.uk/~kjt/research/conformed.html) and Objectives

The objective of this project is to find an alternative way of programming so the program is coded in an more optimised way. This project will focus on Java (particularly when regarding Android mobile applications) to reduce the power needed for the CPU and therefore the device and application uses less battery life.

### Users

The project’s end users are android mobile users who wish to have a device with longer battery life and applications that are less likely to ‘drain’ their device’s battery.

### Objectives

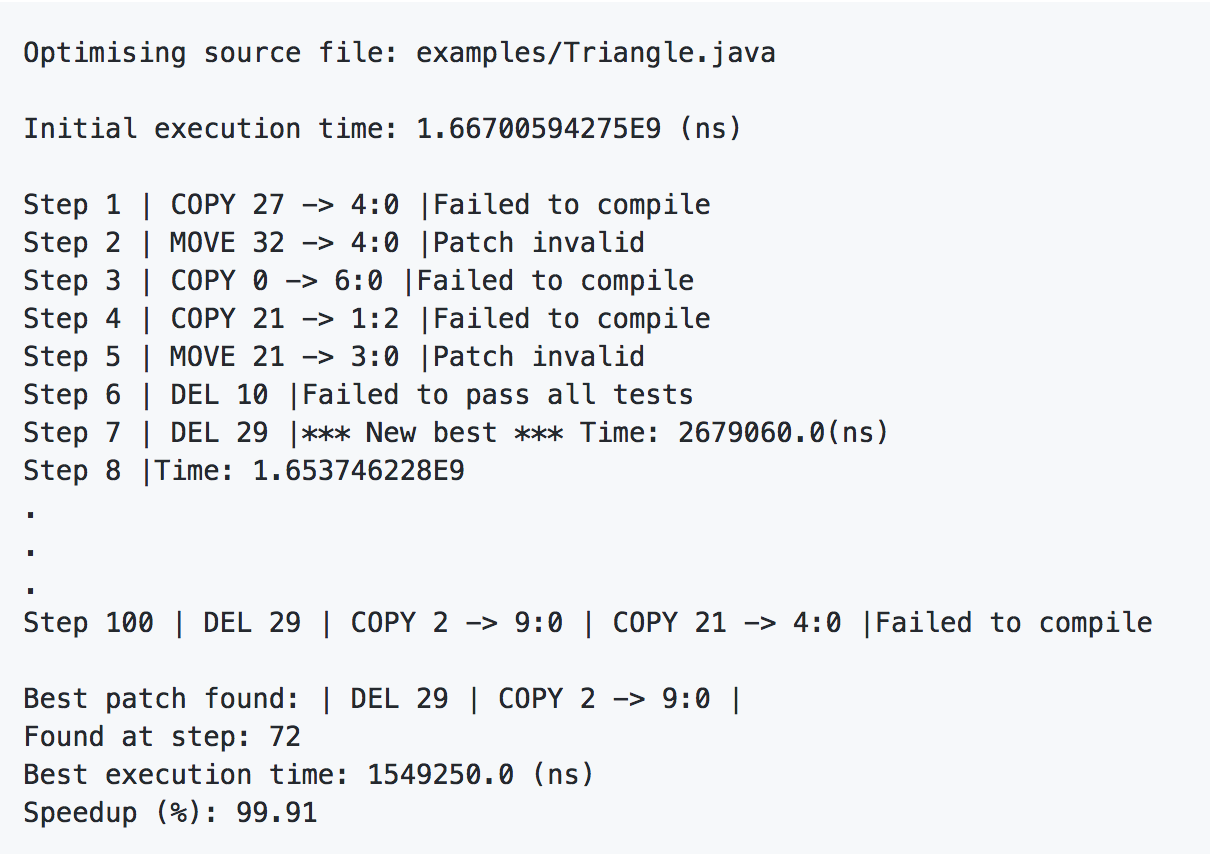
The objectives for the project are:

1. To research the different ways we can optimise Java. Specifically looking at changing operators in code to see its effect and the use of Genetic Improvement to optimise Java.
2. Applying the research from earlier to create some software in a similar vain to the Genetic Improvement in No time (GIN) code that uses genetic algorithms to change source code so it is less execution and energy intensive.
3. To learn about and use the Opacitor and Jalen to prove that the experiments have been successful.
4. To create a piece of software which has a shorter execution time.
5. To create a piece of software that uses less energy to run.

# State-of-The-Art

## GIN

Gin(Genetic Improvement in No time) is a java tool that makes it easier to implement Genetic Improvement(GI). It checks whether a Java class is optimised using a local search and genetic algorithms to add ‘edits’ to code which is a manipulation of code till it finds the most optimised version of the code. GIN manipulates code in three major points which is deletion, copying and replacement.



**Figure 2 GIN. Example Output**

The GIN codebase is an experimental toolkit so allows for experimentation and developing of skills and knowledge which means that they can later apply this skills into an end goal of integrating it in the JVM and compiler.

The GIN code is also one of the very few tools looking at integrating genetic improvement within a java project. The fittest of a genetic algorithm is based on whether the Junit tests are successful and time taken for execution.

Due to the history of GIN, the initial version was written rather quickly so the codebase was not refactored and was a bit complicated when first introduced. The GIN code is going through major refactoring which is making the code base easier to understand.

## Opacitor

Opacitor is a tool which measures the energy consumption of JVM programs. It does this by using a bytecode level model of energy cost. Opacitor ‘can detect small changes in execution profile, down to opcode level’ therefore it records the energy used by a program down to its lowest level. The Opacitor measures energy usage in joules and does not use a CPU as a proxy for energy use like lots of other similar tools have. Therefore using joules and regarding bytecode, increases the accuracy of the readings.

Opacitor is one the first tools of its kind to measure the energy produced of an executing program using Search-Based Software Engineering. Due to the implementation of the Opacitor it can detect the smallest changes in execution therefore recording all the possible energy usage from the entire execution of the program. The opacitor is unaffected by the rest of the computational environment so when it examines the smallest changes these are only changes in the program and does not look at any other environmental factors.

The Opacitor is all very deterministic therefore it will produce the same output from a given input. For example for each class that is put in as an input to the Opacitor leaves an output of how many joules the program used. Opacitor’s results have helped reduce a program’s energy by upto 70% in the best case and 20% in worst case. This deterministic nature means that there is no randomness involved in future development. Similar to a genetic algorithm, as a mutation develops and as generations increase, there is no longer a need for randomness.

The Opacitor is a useful tool however in its implementation, I believe it could be tested more. Currently it has been applied to the Samsung Galaxy S3 mobile phone battery and has not been applied to many others. This means that the variables of hardware of a device or battery has not been included in the results. The opacitor has a potential success rate of decreasing energy usage by 70% but this has only been applied to the Samsung Galaxy S3 so it is unknown whether the same results happen on newer models or different types of phones that also use android applications such as Sony and HTC mobiles.

The Opacitor’s only focus is on the CPU and does not consider other features of a mobile phone that use a large amount of battery such as display, wifi and GPS. The opacitor is exclusive to applications and CPU and does not think of the other feature or how these other features work with applications and what affect this may have on battery.

# Problem description and analysis

## Problem Description

To solve the problem of saving energy when applied to batteries and reducing energy needed by an application. I am aiming to create some software using search based energy optimisation techniques (specifically Genetic Algorithms) to create a piece of software that changes code so it uses less energy and I will measure this improvement using the Opacitor.

## Current Progress

### Predominately I have been doing research into optimisation of Java and Genetic Algorithms through my readings, attending lectures, talking to people in industry and through exploring the GIN and Jcodec codebases. Using this research led me to implement an example algorithm in java which uses a very basic structure of a genetic algorithm with a individual, population and main class. Once I understood genetic algorithms better through this example, I am implementing my own software in a similar vain to GIN in this more simplified genetic algorithm structure. This implementation is a Genetic Algorithm that improves java code that improves execution code. I hope to use what I learnt here and apply it later on to code that saves energy.

## Future Work

After exploring how genetic algorithms work and writing one of my own I’ve started applying the concepts to my own work. Following the progress and ideas of GIN I’ve decided to create a smaller more bespoke version to optimise software for small systems. I then intend to use this as a base and adapt it to use energy efficiency as a measure of fitness, thereby decreasing the energy usage of software that runs on small systems such as mobile phones. This could go a long way to improving battery life therefore improving user experience and helping reduce the impact of mobiles on the environment.

### Execution Time Software

The next steps are to create a Java based software that like GIN improves execution time. I think GIN is a complicated system which is difficult to repurpose and refactor. So by using my research of GIN and genetic algorithms, I wish to adapt an algorithm to serve the same function as GIN. This software aims to be smaller and simpler and allow easier refactoring in the future. Implementing this software will give me a better understanding of how GIN works and developing my software alongside GIN will also set a benchmark to test my program against.

### Energy Saving Software

Once I’ve created the previously mentioned program, I will be able to use what I have learnt to implement a software that can be used to optimise java so it is more energy efficient. Making it this way will mean that the underlying genetic algorithm will be tested and working, it’s only the measure of fitness that has to change. I aim for the code to have a great flexibility as in the future applying different fitness functions could be useful for other cases such as reducing memory. This software will be small and useful for small battery operated systems such as mobile phones but the grounds of the software could be adapted for larger systems where small improvements in efficiency can save large amounts of energy.

References

[1]A. Brownlee, N. Burles and J. Swan, "Search-Based Energy Optimization of Some Ubiquitous Algorithms", *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 3, pp. 188-201, 2017.

[2]M. Negnevitsky, *Artificial intelligence*. Harlow [etc.]: Addison-Wesley/Pearson, 2011.

[3]M. MITCHELL, *An introduction to genetic algorithms*. Cambridge, Mass.: MIT Press, 1996.

[4]S. Luke, *Essentials of Metaheuristics*. [S. l.]: Lulu, 2015.

[5]B. Bruce, J. Petke and M. Harman, "Reducing Energy Consumption Using Genetic Improvement", *GECCO*, 2015.

[6]N. Burles, E. Bowles, A. Brownlee, Z. Kocsis, J. Swan and N. Veerapen, "Object-Oriented Genetic Improvement for Improved Energy Consumption in Google Guava", *Search-Based Software Engineering*, 2015.

[7]N. Burles, E. Bowles, B. Bruce and K. Srivisut, "Specialising Guava’s Cache to Reduce Energy Consumption", 2018.

[8] S. Black and V. &rarr;, "Opacitor: search-based energy optimization of some ubiquitous algorithms #SBSE", *DAASE project blog*, 2018. [Online]. Available: https://daaseblog.wordpress.com/2017/06/01/opacitor-search-based-energy-optimization-of-some-ubiquitous-algorithms-sbse/. [Accessed: 07- Oct- 2018].

[9]D. White, "gintool/gin", *GitHub*, 2018. [Online]. Available: https://github.com/gintool/gin. [Accessed: 07- Oct- 2018].