Dissertation Presentation

1 MIN PER SLIDE

160 WORDS PER SLIDE

Slide 1

* Hi, I’m Anna and for my project I was researching in ways we can change source code to save energy.
* This project is looking as saving energy taken by running programs; \*this is important because we all want longer battery life, and datacentres are killing the planet (or something like that!)
* Firstly I am going to start with some background.

Slide 2

* Genetic Improvement can be used to improve pre-existing software with improvements such as reducing energy consumption, execution time and memory consumption.
* Genetic Algorithms are a form of Genetic Improvement.
* As seen here, there are 5 stages of a genetic algorithm:

1. Initialize population – this process is the creation of a set of individuals which is a population. Each individual is a solution to a problem .
2. Fitness – this process finds out how ‘fit’ an individual is, as in how well suited to solving the problem is this particular individual.
3. Selection – this stage decides what traits are to be kept for the future generations. Traits that are not of a lower fitness are disregarded by the selection stage in the next iteration.
4. Crossover – this stage requires taking genes from the two parents and combining them to make a new individual who may be a more optimised solution to the problem than their parents.
5. Mutation – this stage, we in effect copy some of the individuals from the crossover stage and change them slightly and add those to the new population
6. This then loops

Slide 3

* Earlier I mentioned fitness and I’m going to elaborate further here.
* Fitness is the way we find out if an individual is suited for finding an answer to the problem.
* For this problem there are several notions of fitness:

1. Does the new source code compile?
2. Does the new source code pass the unit tests?
3. What is the energy consumption of the application?

* In order to use all of these fitnesses, I used something called Lexicographical fitness. Whichis the idea that you do not measure the other measures of fitness if one fitness is really high or fails. So in this instance, you do not measure the energy if the individual fails any of their unit tests.
* This is because the fitness of the unit tests were used to find out if the functionality of the code had been comprised. So if the unit tests fail we know that the code does not work the way we want it to.

Slide 4

* This leads us onto the notion of Green Software Engineering.
* This notion was devised by Dr Hindle.
* Green software engineering is a type of software engineering which is more environmentally sustainable, so it takes into consideration the power usage of creating software.
* Research that was conducted by Hindle focuses on software’s interaction in other contexts other than mobile applications. Hindle researches in mobile applications, Data Centres and embedded sensors. Hindle has a proposal to have an energy rating on mobile applications included with the mobile application other functions.
* This type of research is important in our modern times. We now have a technology reliant society and about 12 more years to save the planet.
* The combination of the theories of Green software engineering and genetic improvement lead to the creation of a toolkit that I’ll mention in a slide or two.

Slide 5

* GIN (Genetic Improvement in no time) is a Java tool that makes it easier to implement Genetic Improvement and has a goal “to stimulate development in GI tooling”.
* The Gin application itself optimises a Java class, using a local search to add “Edits” to the source code until a more-optimised version of the code is found
* The GIN toolkit has a fitness that focuses on optimizing code based on whether it passes unit tests and also whether the code can be changed to have a different execution time.

Slide 6

* Most of my project was focused on creating my own version of the GIN toolkit.
* The aptly (not narcissistic) named AnnaGin is a toolkit which tries to find an optimised version of source code by changing the source code till there is a version of the code that uses less energy.
* In order to find out which version of the source code is the most optimised, I used a (pseudo) Genetic Algorithm.
* AnnaGin was made using Java and the spring framework
* I tried to make the toolkit more extensible and flexible than GIN. This was achieved by using spring to easily add new edits and also make it easier to add more forms of fitness so the toolkit can be adapted for many different things.

Slide 7

* The AnnaGin toolkit works like this:

1. The toolkit takes in some source code (that’s written in Java)
2. Gets a group of edits from these options. An edit is a change that can be applied to the code such as removing a like
3. It applies these edits to the source code. This new source code is then called a patch.
4. Measures whether it passes it’s corresponding unit tests and it measures the energy of that source code.

* This makes a individual
* These steps are repeated till it creates a population of a certain size.
* Then all of the above steps are repeated for a certain number of iterations.
* The larger the initial population and the higher the number of iterations, the more successful the genetic algorithm is, at finding a more optimised version of source code.

Slide 8

* Opacitor is a tool which measures the energy consumption of JVM programs.
* The version of the Opacitor I used was the one that measures bytecode. It does this by using a bytecode level model of energy cost; counting the bytecodes executed by a running program and matching these to previously measured quantities of energy consumption.
* Opacitor ‘can detect small changes in execution profile, down to opcode level
* Opacitor is one of the first tools of its kind to measure the energy produced by an executing program using Search-Based Software Engineering. Due to the implementation of the Opacitor it can detect the smallest changes in execution, thereby recording all the possible energy usage from the entire execution of the program.
* It finding small detection made it ideal for this project

Slide 9

* Throughout the project, two important questions were asked: What changes to code can developers make to reduce the program’s energy consumption? And can GI techniques like a GA find a version of the code that uses the least amount of energy?
* The chart here shows the best case scenarios. That means this data is the data that was applied the best and saved the most energy for each type of edit.
* These experiments where conducted by applying one edit of one type to the triangle program which was the tiny program shown a few slides back.

Slide 10

* As stated earlier, As you increase the size of the initial population and number of iterations, the average fitness of the end population decreases.
* Therefore the GA gradually gets better at finding a solution that uses the least amount of energy.
* The result here were compared to a random search. In comparison, the Genetic Algorithm did infinitely better at finding a result than the random search in a shorter time period with the same sized initial population.

Slide 11

* The major idea behind this research is that you can apply this to larger things like data centers where only saving a couple of joules of energy can possibly save millions of dollars.
* The toolkit could be remade or changed so it can optimise programs in other languages.
* Toolkit could be adapted to optimise code on other notions of fitness such as reducing memory or finding code without bug fixes.
* There are lots of other possible edits that could be made. As part of the project I conducted experiments on whether iterative verses recursive loops and their energy consumption. This research could be made into an edit so it can change a recursive loop into iterative.

Slide 12

* Thank you so much for listening.
* Does anyone have any questions?

Possible Questions