Assignment 2: Optimisation of non-functional properties

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

Due date: 11:59pm, 22 September 2017, Weight: 10.0 % of the course

Preparation

This assignment is going to be exciting: actual hands-on optimisation of non-functional properties!

In this course, we have already talked about the optimising code for energy consumption, and we have talked about multi-objective optimisation.

First, it is mandatory that you do some reading:

- `sbse-assignment02-paper.pdf` is a copy of the FSE 2015 paper "Optimizing energy consumption of GUIs in Android apps: a multi-objective approach", https://myuni-canvas.adelaide.edu.au/files/1819780/download?download_frd=1
- `sbse-assignment02-slides.pdf` are slides to provide a gentle introduction to the topic, https://myuni-canvas.adelaide.edu.au/files/1819781/download?download_frd=1

There is no need to understand everything, however, we strongly recommend to spend maybe two to three hours as a team with reading both PDFs and looking up terms that you have not heard of before. If something remains really unclear, then please ask via the discussion forums.

The first assignment has been relatively theoretical. This second assignment is largely hands-on and it will involve programming in Java. Do not be afraid: the necessary Java concepts are super basic. If your Java skills or programming skills in general are a bit rusty, then you might want to spend a couple of hours to remind yourself how to write basic code, e.g., access arrays, use loops, use functions, etc.

Lastly, download the following two files:

- Nexus 6 experimental data, https://myuni-canvas.adelaide.edu.au/files/1820247/download?download_frd=1

- The target application with two graphical user interfaces (inside the Calculator package), and an optimiser app (called GuiOptimiser) that can serve as a starting point for the optimisation. The optimiser will run the target app in full-screen mode, take a screenshot of its graphical user interface, and save it to disk. This was tested under Windws/Linux/MacOS. Download link: `https://myuni-canvas.adelaide.edu.au/files/1820752/download?download_frd=1`

In case you cannot take screenshot reliably, try increasing the following value in guioptimiser.GuiOptimiser:
`TARGET_APP_RUNNINGTIME = 1000; // try 5000 or even longer.` If you encounter technical problems, please post your problem with a description of what you have tried on the discussion forums. This way, we can help not only you but also others.

1 Overview

Assignments should be done in groups consisting of two (2) students. If you have problems finding a group partner use the forum to search for group partners or contact the lecturer.

2 Assignment

Exercise 1 *Determine the surrogate function (20 points)*

Preparation

Although it could be quite fun for you to perform the optimisation on actual Android phones, we do not want you to go through some of the troubles that we have had to deal with in the past — see the lecture on deep parameter optimisation for details.

Instead of using the actual device, you will use a surrogate function that will provide you with reasonably accurate fitness values. To help you with this, Mahmoud has run some experiments. On a Nexus 6, he has recorded the total system's charge consumption[1] when the entire screen was set to various colours (for about 30 minutes each time): white, black, red, green, . . . , while the actual brightness of the screen was set to its maximum. He has done this by sampling (at a rate of about 4 Hz) the amount of charge that is left in the phone.

Remember the comment from the lecture: the Nexus 6 has a screen with organic light-emitting diode (OLED) technology. This means that different colours can consume different amounts of charge.

Have a look inside the spreadsheet with the experimental data. This spreadsheet contains five tables, but only the table named "screenOn" is of interest to us. This table lists all the samples taken over time, and the plot shows the charge consumed of the entire device when various colours were displayed. Black can be seen as a base case, that is, it shows the

[1]Simply speaking: Energy = Charge X Voltage, however, we want to keep it simple here.

charge consumed by the device when nothing is shown on the screen.[2] You can see that red and green use similar amounts of charge, i.e., the draw different currents. Note how showing red and green at the same time make yellow, however, the charge consumption of yellow is not the same as the sum of the energies consumed by both red and green independently.

Part 1

Determine the average charge consumed for the colours red, green, and blue.

Calculate the average charge used per hour when considering only the first 10, the first 100, and the first 1000 samples, and when considering the entire duration of the experiment. You will see that the average consumption varies slightly depending on the duration of the experiment. To to this, use the columns named `charge_used_accumulated`[3] (the units are nAh for nano Ampere hours) and `system_time_in_ms`.

The unit that you use should be mA (milli Ampere), and you should use the scientific notation to report the results.

In total, you need to provide 12 numbers for the three colours and the four sample set sizes.

Notes:

- Do not forget to deduct the charge consumed for black.

Part 2

Write a function `calculateChargeConumptionPerPixel` in Java that calculates the charge consumption (== current) of a single pixel. This function should take three integers (in the range 0...255) as parameters: the pixel's red, green, and blue values.

Integrate this function into our provided framework so that you can read image files and calculate the charge consumption — base your calculations on the per-pixel consumption of the Nexus 6 (based on the maximum number of samples available in the spreadsheet), even though your images might have a different resolution.

To demonstrate that your code works, calculate the charge consumption for three images (or screenshots) of your choice.

For this part, also submit these three images, the number of pixels in the image, and the charge consumption that you calculated.

Notes:

- The Nexus 6 has a screen resolution of 1,440 x 2,560 = 3,686,400 pixels.
- In case you got stuck in Part 1, you can use the following consumption rates when the entire screen shows a particular colour: red 120mA, green 140mA, blue 240mA.

[2]"nothing is shown" here means that "every pixel is set to black". Still, the screen is on. It is possible to "deactivate the screen" in Android, but this also causes lots of changes to the scheduling of tasks — which is something that we (and you) are not interested in here.

[3]This is based on `CHARGE_COUNTER_EXT` with a 8nAh resolution, see https://source.android.com/devices/tech/power/device.

- Since you will be working with images, here are some pointers at background information, including code and libraries:
  https://www.tutorialspoint.com/java_dip/understand_image_pixels.htm
  https://stackoverflow.com/questions/6524196/java-get-pixel-array-from-image
  https://stackoverflow.com/questions/22391353/get-color-of-each-pixel-of-an-image-using-bufferedimages
  https://www.dyclassroom.com/image-processing-project/how-to-get-and-set-pixel-value-in-java

Exercise 2 *Single-objective minimisation of charge consumption (40 points)*

Part 1 - Setup

Mahmoud's provided framework does not keep track of the best colour settings. Extend his code to implement a proper random search that outputs the best colour settings found. Keep in mind that we want to minimise charge consumption, not maximise it.

In addition, design and implement two additional and different optimisation algorithms. This can be a hill-climber, a simulated annealing, and a genetic algorithm.

To make things a little bit more realistic, let us assume a customer brings along the following requirement that needs to be satisfied by all colour schemes:

> The Euclidean distance between the colours (in the three-dimensional colour-space) for the text in the text field and the text field itself should be at least 128.

What does this do? Well, it is an attempt to ensure that the text in the text field is still readable. In particular, jTextField1 is the background color of jTextField1, and jTextField1TextColor is the foreground (text) color of jTextField1

Only colour configurations that do not violate this constraint should be evaluated via a screenshot. Do not waste resources on evaluating unacceptable solutions.

Part 2 - Experiments

Now, optimise both target user interfaces with all three algorithms 10 times. The computational budget is 1000 screenshots for each run.

For each target user interface for for each algorithm report the following:

- First, for each of the 10 independent runs, record the charge consumption of the best configuration after 10 screenshots (and after 100 screenshots, and after 1000 screenshots). Second, average these numbers for the 10 independent runs.
- Submit the best final configuration found and the corresponding screenshot.

It is voluntary to use fancy visualisations (e.g. box plots) to show the performance.

Notes:

- To represent a solution, it might be natural to consider integer arrays `int`[].

- As the experiments can generate a lot of data, you might want to include a function that deletes the screenshot as you as you have calculated its charge consumption.
- In case you find it difficult to process the screenshots produced in the PNG format, then you can change the format to BMP in guioptimiser.Capture.

Exercise 3 *Multi-objective optimisation (40 points)*

Part 1 - MOEA Framework

MOEA Framework is an object-oriented Java-based framework for multi-objective optimisation with metaheuristics, `http://moeaframework.org/`. We will use it so that you do not have to design your multi-objective optimiser from scratch.

Download it and run NSGA-II and MOEA/D on the four different problems ZDT1, ZDT2, DTLZ3, and DTLZ4. ZDT1 and ZDT2 are problems with 2 objectives. The DTLZ problems can be varied in the number of objectives—for this assignment, you need to set the number of objectives to 3.

Run each algorithm on each problem 10 times, with a population sizes of 20 and 100, and with a computation budget of 10,000 evaluations → 16 combinations that are to be run 10 times.

Report for each combination:

- The final covered hypervolume: average and standard deviation of the 10 repetitions.
- Show the final populations as scatter plots in the objective space.

Ideally, do not submit to us 160 plots, but put them together in a way that you think supports the comparison of the different combinations.

Notes:

- MOEA Framework API: `http://moeaframework.org/javadoc/index.html`
- MOEA Framework user manual (Chapters 2, 3, and 5 are the most important ones): `https://sourceforge.net/p/moeaframework/bugs/_discuss/thread/12d4a637/4a31/attachment/MOEAFramework-2.1-ManualFixed.pdf`
- MOEA Framework examples: `http://moeaframework.org/examples.html`

General comment. The purpose of contrasting two quite different population sizes is to show that parameter choices can matter depending on the algorithm and problem. For a while, the topic of "automated algorithm configuration" (AAC) has been quite hot, where algorithms tune other algorithms, and with the relevant papers being cited hundreds of times. What is AAC about? In short, it is about tuning an algorithm and all its parameters and switches such that the resulting configuration works well on a set of (typically unseen) problem instances. Slightly newer is the idea of configuring an algorithm on-the-fly either at the beginning ("per-instance-configuration") based on information that is easily available, or during a run (going into the direction of "hyperheuristics") based on observed features during the run. In future Evolutionary Computation Courses, we might dig into this a lot more — and there is a super tiny chance of AAC appearing in Assignment 3 here — but for now this is beyond the scope of this Assignment 2.

Part 2 - Multi-objective optimisation of charge consumption

Imagine a customer who comes to you with an existing system. She asks you to minimise a particular property (here: charge consumption due to the colour profile), but who is also interested in seeing the trade-offs between deviating from the current profile and the charge saved.

In particular, the deviation is measured as: the sum of of Euclidean colour differences for each pixel. This deviation is of course always ≥ 0.

Step 1) Since we do not have a particular client, you need to define your own reference colour scheme. You can define something that you think is visually pleasant, or just take a random one. The only limitation is that not all colours are the same. Do this for both user interfaces.

Step 2) Implement in MOEA Framework the charge consumption and the deviation from your target at a two-objective problem. Both objectives are to be minimsed.

Pick one multi-objective algorithm (for example, NSGA-II or SPEA2). Run it once with a total budget of 10,000 screenshots, and with a population size $\mu = 20$.

Report on the outcomes (separate for each target user interface):

- Show as a scatter plot (in 2D) the distribution of the first generation and of the last generation.
- Provide all 20 screenshots (or fewer, see note below!) of the final population. Also add them to a scatterplot so that we can easily see the trade-offs.

Of course, also submit your reference screenshots.

An important note regarding those 20 screenshots: it can happen that the algorithm returns a set of configurations with fewer than 20 solutions. How come? Well, many multi-objective optimisation algorithms (or frameworks that provide them) decide that they only return as the final solution set the set of non-dominated solutions. This means that if a solution is dominated (for example, because it is worse in all objectives than another solution in the solution set), then this will be omitted in the end. For you, this means that your final solution sets can have anywhere between 1 and 20 non-dominated solutions, so you might be sending us fewer than 20 screenshots.

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 solutions for theoretical assignments
- all source files (if you created any)
- descriptions as required in the statement of the exercises

- 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

Note: there will be a progress presentation session for this assignment. This is a big opportunity for you to get feedback on your progress, and to make last adjustments for the final submission.