Lab 1: Genetic Algorithms[1]

Released: Wednesday 31st January, 2018
Deadline: February 2, 2018
Weight: 10 % for 06-27819, or 11.25 % for 06-27818 %

This coursework carries 10 % towards your final mark if you follow 06-27819, and 11.25 % if you follow 06-27818. The marks are shown after each exercise. You need to implement one program that solves Exercise 1-5 using any programming langauge. In Exercise 6, you will run a set of experiments and describe the result using plots and a short discussion.

You need to submit one zip file with the name `niso1-abc123.zip` to Canvas, where you replace `abc123` with your username. The zip file should contain a directory with the following files:

- the source code for your program
- a Dockerfile (see the Appendix for instructions)
- a PDF file for Exercise 6.

In this lab, we will implement a simple non-elitist genetic algorithm using mutation, crossover, and tournament selection. The pseudo-code of the algorithm is given below. We will investigate the performance of this algorithm on a simple fitness function, showing how it depends on the problem size, the mutation rate, the population size, and the tournament size.

Algorithm 1 Genetic Algorithm

Require: Problem size $n \in \mathbb{N}$
Require: Population size $\lambda \in \mathbb{N}$ and tournament size $k \leq \lambda$
Require: Fitness function $f : \{0,1\}^n \to \mathbb{R}$
for $i = 1$ to λ **do**
  $P_0(i) \sim \text{Unif}(\{0,1\}^n)$
end for
for $t = 0 \ldots$ until terminal condition satisfied **do**
  for $i = 1$ to λ **do**
    $x = \texttt{tournament\_selection}(P_t)$
    $y = \texttt{tournament\_selection}(P_t)$
    $P_{t+1}(i) = \texttt{crossover}(\texttt{mutation}(x), \texttt{mutation}(y))$
  end for
end for
return P_t

[1] Revised: Wednesday 31st January, 2018 at 09:31.

Exercise 1. *(10 % of the marks)*

Implement a mutation operator for bit-strings with mutation rate χ/n, with $\chi \in [0, n]$ and $n \in \mathbb{N}$. (The Greek letter χ is pronounced chi.*) Given an input bitstring $x \in \{0,1\}^n$, the output should be a bitstring $y \in \{0,1\}^n$ such that $\Pr(y_i = x_i) = 1 - \chi/n$, independently for each $i \in \{1, \ldots, n\}$.*

Input arguments:

- `-bits_x` *the input bitstring x*
- `-chi` *the specification of mutation rate (χ/n, where n is length of bitstring)*
- `-repetitions` *the number of repetitions*

Output:

- *the bitstring after mutation*

Example:

```
[pkl@phi ocamlec]$ niso_lab1 -question 1 -bits_x 00000 -chi 2.5 -repetitions 4
01010
10000
01100
11100
```

Exercise 2. *(10 % of the marks)*

Implement the uniform crossover operator for bit-strings. Given two input bitstrings $x \in \{0,1\}^n$ *and* $y \in \{0,1\}^n$*, the output should be a bitstring* $z \in \{0,1\}^n$ *such that* $\Pr(z_i = 0) = 1/2$*, if* $x_i \neq y_i$*, and* $\Pr(z_i = x_i) = 1$ *otherwise, independently for each* $i \in \{1, \ldots, n\}$*.*

Input arguments:

- `-bits_x` *the input bitstring* x
- `-bits_y` *the input bitstring* y
- `-repetitions` *the number of repetitions*

Output:

- *the crossover products, one line per repetition*

Example:

```
[pkl@phi ocamlec]$ niso_lab1 -question 2 -bits_x 00000 -bits_y 11111 -repetitions 4
10111
11110
01111
10001
```

Exercise 3. *(10 % of the marks)*

Implement the OneMax: $\{0,1\}^n \to \mathbb{R}$ *fitness function, which is defined as*

$$\textsc{OneMax}(x) := \sum_{i=1}^{n} x_i.$$

Input arguments:

- `-bits_x` *input bitstring*

Output:

- *The fitness value of the input bitstring.*

Example:

```
[pkl@phi ocamlec]$ niso_lab1 -question 3 -bits_x 01000
1
```

Exercise 4. *(10 % of the marks)*

Implement tournament selection with tournament size k, as described by the following algorithm.

Algorithm 2 Tournament Selection

Require: Population size $\lambda \in \mathbb{N}$ and tournament size $k \leq \lambda$
Require: Fitness function $f : \{0,1\}^n \to \mathbb{R}$
Require: Population $P = (y^{(1)}, \ldots, y^{(\lambda)})$ with $y^{(i)} \in \{0,1\}^n$ for all $i \in [\lambda]$
1: Sample uniformly at random (with replacement) a set S of k elements from $\{1, \ldots, \lambda\}$.
2: $j \sim \text{Unif}\left(\arg\max_{i \in S} f(y^{(i)})\right)$
3: **return** $y^{(j)}$

Your program should have the following input/output behaviour. Use OneMax *as fitness function.*

Input arguments:

- `-k` *tournament size*
- `-population` *bitstrings*
- `-repetitions` *the number of repetitions*

Output:

- *the selected bitstring*

Example

```
[pkl@phi ocamlec]$ niso_lab1 -question 4 \
    -population "0000 1111 1110 1100 1000 0000" -k 2 -repetitions 4
1111
1000
1111
1110
```

Exercise 5. *(30 % of the marks)*

Implement the simple genetic algorithm in Algorithm 1.

Your program should have the following input/output behaviour, using OneMax *as fitness function.*

Input arguments:

- `-n` *bitstring length*
- `-chi` *specification of mutation rate*
- `-k` *tournament size*
- `-lambda` *population size*
- `-repetitions` *the number of repetitions*

Output:

The output should be one line per repetition, where the following values are shown separated by the tab character.

```
n    chi    lambda    k    t    fbest    xbest
```

where

- `t` *number of generations until optimum found*
- `fbest` *fitness of fittest individual*
- `xbest` *bitstring of fittest individual*

Example:

```
[pkl@phi ocamlec]$ niso_lab1 -question 5 -chi 0.2 -n 10 -lambda 10 -k 2 -repetitions 4
10  0.2   10   2   4   10.000000 1111111111
10  0.2   10   2   8   10.000000 1111111111
10  0.2   10   2   10  10.000000 1111111111
10  0.2   10   2   4   10.000000 1111111111
```

Exercise 6. *(30 % of the marks)*

The runtime of an evolutionary algorithm is defined as the number of function evaluations (i.e., number of generations multiplied with the population size λ) before the algorithm discovers the optimal solution for the first time. Run experiements with the genetic algorithm on OneMax *where you investigate the following relationships*

- *runtime vs mutation rate χ, for $n = 200$ and $0 < \chi < 3$ and $\lambda = 100$*
- *runtime vs problem size n, for $10 < n < 200$ and $\chi = 0.6$ and $\lambda = 100$*
- *runtime vs population size, for $10 \leq \lambda \leq 1000$ for $n = 200$ and $\chi = 0.6$*
- *runtime vs tournament size k, for $2 \leq k \leq 5$ and $\chi = 0.6$ and $\lambda = 100$*

Repeat each experiment 100 times. Show the results as boxplots (e.g., using the statistical software R), and discuss the results. Note that some of the experiments may not terminate within reasonable time. You therefore need to specify a time-out value, and stop the algorithm after the specified number of generations.

A. Docker Howto

You can use any programming language to solve Exercise 1-5. However, to submit your solution, you need to prepare a so-called Docker image which provides an isolated environment for running your code. By submitting your solution as a Docker image, you can be sure that all libraries and tools are set up correctly to compile and run your program. Furthermore, we can be sure that when testing your program, it does not interfere with our machines.

A Docker image is typically built on top of an existing base image, e.g., a Linux distribution. We have prepared a Docker image (pklehre/niso-lab1) which is built on top of Debian Linux and which contains a program to test your code. To build your Docker image, you need to extend this base image, install any necessary software (using the apt-get package manager in Debian), and compile the code if necessary. After building the container, you can run the container to evaluate your solution. Once you are happy with your solution, you can submit your Doker image on Canvas.

Follow these steps exactly to build, test, save, and submit your Docker image. Please replace *abc123* in the text below with your username.

1. Install Docker CE on your machine from the following website:
   `https://www.docker.com/community-edition`

2. Copy the PDF file from Exercises 6 and all required source files, and/or bytecode to an empty directory named niso1-abc123 (where you replace abc123 with your username).

```
mkdir niso1-abc123
cd niso1-abc123/
cp ../exercise6.pdf .
cp ../abc123.py .
```

3. Create a text file `Dockerimage` file in the same directory, following the instructions below.

```
# Do not change the following line. It specifies the base image which
# will be downloaded when you build your image. We will release a new
# base image for each lab.

FROM pklehre/niso-lab1

# Add all the files you need for your submission into the Docker image,
# e.g. source code, Java bytecode, etc. In this example, we assume your
# program is the Python code in the file abc123.py. For simplicity, we
# copy the file to the /bin directory in the Docker image. You can add
# multiple files if needed.

ADD abc123.py /bin

# Install all the software required to run your code. The Docker image
# is derived from the Debian Linux distribution. You therefore need to
# use the apt-get package manager to install software. You can install
# e.g. java, python, ghc or whatever you need. You can also
```

```
# compile your code if needed.

# In this example, we install Python and Numpy. Code to install Java
# is commented out.

RUN apt-get update
# RUN apt-get -y install default-jre default-jdk
RUN apt-get -y install python
RUN apt-get -y install python-numpy

# The final line specifies your username and how to start your program.
# Replace abc123 with your real username and python /bin/abc123.py
# with what is required to start your program.

CMD ["-username", "abc123", "-submission", "python /bin/abc123.py"]
```

4. Build the Docker image as shown below. The base image `pklehre/niso-lab1` will be downloaded from Docker Hub

```
docker build . -t niso1-abc123
```

5. Run the docker image to test that your program starts. A battery of test cases will be executed to check your solution.

```
docker run niso1-abc123
```

6. Once you are happy with your solution, compress the directory containing the Dockerfile as a zip-file. The directory should contain the source code, the Dockerfile, and the PDF file for Exercise 6. The name of the zip-file should be `niso1-abc123.zip` (again, replace the abc123 with your username).

7. Submit the zip file `niso1-abc123.zip` on Canvas.