Evolutionary Computation 2006: Home problems, set 2

General instructions. READ CAREFULLY!

Problem set 2 consists of three parts. Problem 2.1 is mandatory, the others voluntary (but check the requirements for the various grades on the web page). After solving the problems, collect your answers and your programs in *one* zip file which, when opened, generates one folder for each problem (e.g. Problem 2.1, Problem 2.2 etc.) in the assignment. Make sure to keep copies of the files that you hand in!

You should provide a brief report in the form of a PDF, PS or text file.

In *all* problems requiring programming, use Matlab (v.6 or v.7). The *complete* Matlab program for the problem in question (i.e. all source files) must be handed in, collected in the same folder. In addition, *clear* instructions concerning how to run the programs *should* be given in the report. It should *not* be necessary to edit the programs, move files etc. Programs that do not function or require editing to function will result in a deduction of points.

The maximum number of points for problem set 2 is 15. Incorrect problems will *not* be returned for correction, so please make sure to check your solutions and programs carefully before e-mailing them to `mattias.wahde@chalmers.se`.

You may, of course, discuss the problems with other students. However, each student *must* hand in his or her *own* solution. In obvious cases of plagiarism, points will be deducted from all students involved. Don't forget to write your name and civic registration number on the front page of the report!

NOTE: Please make sure to follow the instructions above, as a failure to do so may result in a deduction of points!

Deadline: 20061020

Problem 2.1, 5p, The traveling salesperson problem

The traveling salesperson problem (TSP) has many applications in e.g. network routing and placement of components on circuit boards. In this problem, you will solve the TSP in a case where the cost function is simply taken as the length of the route. Write a GA that can search for the shortest route between N cities, using permutation coding for the path. Use an encoding method such that the chromosomes consist of lists of integers determining the indices of the cities (Hint: use the command `randperm` in Matlab to generate such chromosomes). Examples of five-city paths starting in city 4 are e.g. (4,3,1,2,5), (4,1,5,2,3), (4,5,1,2,3) etc. The first chromosome thus encodes the path $4 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 4$. The fitness should be taken as the inverse of the route length (calculated using the ordinary cartesian distance measure, i.e. *not* the city-block distance measure). The program should always generate syntactically correct routes, i.e. routes in which each city is visited once and only once until (NOTE!), in the final step, the tour ends by a return to the starting city.

Specialized operators for crossover and mutation are needed in order to ensure that the paths are syntactically correct. Use order crossover as described in problem 5.3b. Your program *must* contain a separate function `order_crossover` with *exactly* the following interface

```
function [c1new, c2new] = order_crossover(c1,c2);
```

where `c1`, `c2`, `c1new`, and `c2new` are chromosomes (paths). Your crossover function will be tested, assuming that the interface looks exactly as above. For mutations use the swap mutations defined in problem 5.3a.

Solve the following problems:

(a) In the TSP, paths that start in different cities but run through the cities in the same order are equivalent (in the sense that the path length is the same). Furthermore, paths that go through the same cities in opposite order are also equivalent. Thus, for example, the paths $(1, 2, 3, 4, 5)$ and $(2, 3, 4, 5, 1)$ are equivalent, as are $(1, 2, 3, 4, 5)$ and $(5, 4, 3, 2, 1)$. Paths that are *not* equivalent are called *distinct paths.* How many distinct paths are there in the general case of N cities?

(b) Use your program to search for the shortest possible path between the cities whose coordinates are given in the file

```
www.me.chalmers.se/~mwahde/courses/ec/2006/TSPcities1_2.m
```

This file contains a 50×2 matrix with the coordinates (x_i, y_i) for city i, $i = 1, \ldots, 50$. In addition to the full matlab program, send also the shortest path you have found, in electronic format, i.e. in a text file or a matlab file (.mat), containing a vector with the city *indices* for the path in question. Note that the indices *should* be in the interval $[1, 50]$, *not* $[0, 49]$. For example, a path may be given as

`bestpath = [4 7 11 39 50 41 3 ...` etc.

The length of the path will be tested using the vector that you provide. Finally, in your report, draw the shortest path, and specify its length. For full points, it is required that the length of your shortest path should be less than 150 length units.

Problem 2.2, 5p, The 3-parity problem

Exclusive or (XOR) is a commonly used logical operator, which takes two inputs and for which the output y is equal to 1 if exactly one of the inputs x_1 or x_2 is equal to 1, and 0 otherwise. Thus, the **truth table** for XOR is given by

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

XOR can be generalized to n inputs, and the resulting Boolean function is called an $n-$**parity function.** These functions give the output 1 if an odd number of inputs are equal to 1, and 0 otherwise. Parity functions can be represented by feedforward neural networks (FFNNs), containing n inputs, n neurons in the hidden layer, and one output. For a given value of n, the number of parameters (weights and biases) in such a network is equal to $(n+1)^2$.

Using a GA of your choice, evolve a discrete-time FFNN with 3 inputs, 3 hidden neurons, and one output, which generates the 3-parity function. The neurons in the FFNN should use the squashing function

$$\sigma(z) = \frac{1}{1+\mathrm{e}^{-cz}}, \tag{1}$$

for some suitable value of c (around 1). Let the fitness be the inverse of the mean-square deviation Δ between the desired output d and the actual output y, i.e. set $f = 1/\Delta$, where

$$\Delta = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(d(i) - y(i)\right)^2}, \tag{2}$$

where $m = 8$ for $n = 3$. Set the cutoff fitness f_c (for the EA runs) such that runs are terminated when Δ is equal to 0.01. (In case the evolved networks fail to generate the correct output[1] even when the fitness cutoff is reached, you may wish to use a different fitness measure, which is more focused on the *worst* output. For example, you may use the p^{th} root (with $p > 2$) of the average of the p^{th} powers of the differences $d(i) - y(i)$).

In your report, describe the encoding scheme (e.g. binary, real-number) that you have used and specify the operators that have been used for selection, crossover, and mutation, as well as the values of the various GA parameters (e.g. p_c, p_{mut}, population size etc.) and the value of c (see Eq. (1)). Note: for values of c around 1, a suitable range for the weights (and biases) is approximately $[-10, 10]$. In addition to your GA (i.e. the complete Matlab program), you should also provide (in your report) a plot over the average and maximum fitness as a function of the number of evaluated individuals. Furthermore, you should provide a test program where the user may enter three bits and then obtain the output from your *best* network. The interface to the test program should be exactly as follows

```
output = testnetwork(x1,x2,x3),
```

where `x1`, `x2`, `x3` are the three input bits. Encode the best network directly into the test program, so that the program runs directly, without the use of any additional input files etc.

[1]Definition of correct output: if, for a given input signal, the desired output is equal to 0, an output of 0.02 or less will be considered correct. If instead the desired output is equal to 1, an output of 0.98 or more will be considered correct.

Problem 2.3, 5p, Function fitting using LGP

Consider a case in which a data series has been generated from a function of the form

$$g(x) = \frac{a_0 + a_1 x + a_2 x^2 + \ldots + a_p x^p}{b_0 + b_1 x + b_2 x^2 + \ldots + b_q x^q}. \tag{3}$$

The values of p and q, as well as the constants a_i and b_i are unknown, and should be inferred from the data series using LGP. Start by writing a general LGP program, with M variable registers, N constant registers, and the operator set $\{+, -, \times, /\}$. The program should evolve linear chromosomes using tournament selection, two-point crossover (see Handout 6), and mutations. The structure of the chromosomes should be of the kind illustrated in Handout 6, i.e. such that each instruction is defined by four numbers. The first number (in a given instruction) should encode the operator and the second number should encode the destination register. The third and fourth number should encode the two operators. The data set, consisting of values of the function $y = g(x)$ for various values of x, is contained in the file

`www.me.chalmers.se/~mwahde/courses/ec/2006/FunctionData2.3.txt`

The evaluation of a chromosome should be done as follows: For each data point (x_k, y_k), place the value of x_k in the first variable register (r_1), and set the contents of the other variable registers to 0. Next, execute the sequence of instructions contained in the chromosome, and take the final value contained in r_1 as the output, i.e. as the estimate $\hat{y}_k$ of y_k. When all data points have been considered, form the total error as

$$e = \sqrt{\frac{1}{K} \sum_{k=1}^{K} \left(\hat{y}_k - y_k\right)^2}, \tag{4}$$

where K is the number of data points. Finally, set the fitness value as $f = 1/e$.

You may use as many variable registers and constant registers as you like. Note that the values of the constant registers should be set once and for all, before the evaluation of chromosomes begins. Run your program, and try to determine the function $g(x)$. The function should be specified *as in Eq. (3) above* (but with numerical values for all constants) in the report. Just providing the best chromosome is *not* sufficient. Also, make sure to send the complete Matlab program. Note: Points may be given even if the function that you find does not correspond exactly to the correct one.

Hint: In order to cope with variable-length chromosomes, you may wish to use the `struct` concept in Matlab, see e.g. the help files in Matlab (`>> help struct` etc.). You can also download a simple example on the course web page.