Evolutionary Computation
Practical Assignment 1

1 Fitness functions

We will look at experimental results of Genetic Algorithms (GAs) on four artificial functions in order to gain some insight in the convergence behavior of GAs. All 4 functions are defined over the binary domain. The goal is to find the binary vector that maximizes the function value; in each case the optimal solution is the string of all ones. We assume a stringlength $\ell = 100$.

1. Counting Ones Functions:

   (a) Uniformly Scaled Counting Ones Function:

   $$x_i \in \{0,1\} : CO(x_1 \dots x_\ell) = \sum_{i=1}^{\ell} x_i$$

   (b) Linearly Scaled Counting Ones Function:

   $$x_i \in \{0,1\} : SCO(x_1 \dots x_\ell) = \sum_{i=1}^{\ell} i x_i$$

2. Trap Functions:

   $$TF(x_1 \dots x_\ell) = \sum_{j=0}^{\frac{\ell}{k}-1} B(x_{jk+1} \dots x_{jk+k})$$

   with:

   $$B(x_1 \dots x_k) = \begin{cases} k & \text{if} \quad CO(x_1 \dots x_k) = k \\ k - d - \frac{k-d}{k-1} CO(x) & \text{if} \quad CO(x_1 \dots x_k) < k \end{cases}$$

   (a) Deceptive Trap Function: $k = 4,\ \ d = 1$

Number of Ones	4	3	2	1	0
Fitness Value	4	0	1	2	3

   (b) Non-deceptive Trap Function: $k = 4,\ \ d = 2.5$

Number of Ones	4	3	2	1	0
Fitness Value	4	0	0.5	1.0	1.5

Both Trap functions consist of 25 concatenated subfunctions of length $k = 4$, each having an optimum at 1111 and at 0000. The overall function has therefore $2^{25} - 1$ local optima and 1 global optimum (= the string of all ones). The difference between the two trap functions is the difference d between the values of the two optima at each substring. As a result of this fitness difference the first trap function is so-called, fully deceptive, while the second is not. Deceptive functions are functions where the lower-order schema fitness averages that contain the local optimum 0000 have a higher value than the lower-order schema fitness averages that contain the global optimum 1111 in each subfunction.

(a) Deceptive Trap Function:

$$\begin{cases} F(111*) = \frac{4+0}{2} = 2 \\ F(000*) = \frac{3+2}{2} = 2.5 \end{cases}$$

$$\begin{cases} F(11**) = \frac{4+0+1}{4} = 1.25 \\ F(00**) = \frac{3+2.2+1}{4} = 2 \end{cases}$$

$$\begin{cases} F(1***) = \frac{4+0+3.1+2}{8} = 1.125 \\ F(0***) = \frac{3+3.2+3.1+0}{8} = 1.5 \end{cases}$$

(b) Non-deceptive Trap Function:

$$\begin{cases} F(111*) = \frac{4+0}{2} = 2 \\ F(000*) = \frac{1.5+1}{2} = 1.25 \end{cases}$$

$$\begin{cases} F(11**) = \frac{4+0+0.5}{4} = 1.125 \\ F(00**) = \frac{1.5+2(1)+0.5}{4} = 1 \end{cases}$$

$$\begin{cases} F(1***) = \frac{4+0+3(0.5)+1}{8} = 0.813 \\ F(0***) = \frac{1.5+3(1.0)+3(0.5)+0}{8} = 0.75 \end{cases}$$

The Trap functions can be *tightly linked* or *randomly linked*.

Tightly linked means that the 25 subfunctions are placed adjacent to each other on the binary string. Randomly linked means that the 100 bits are placed at a random position on the binary string.

2 Genetic algorithm

Selection. We use a generational GA applying tournament selection (tournament size $s = 2$). The offspring population competes against the previous generation (= population elitism). More specifically, assume we have a population of size N at generation T. The 1st solution competes against the 2nd to be selected as parent, the 3rd competes against the 4th, and so on. This results in $\frac{N}{2}$ solutions selected as parent. After randomly shuffling the population we repeat the selection step to select the other $\frac{N}{2}$ parents. Now the 1st and 2nd parents are recombined or mutated to create two offspring solutions, the 3rd and 4th are recombined or mutated, and so on. After evaluating the N offspring solutions, we select the best N solutions from the population at generation T and the offspring population to form the population at generation $T + 1$.

Crossover. Recombination is either uniform crossover or 2-point crossover.

Mutation. First we decide how many bits will be mutated: with probability $\frac{1}{2}$ flip 1 bit, or with probability $\frac{1}{4}$ flip 2 bits, or with probability $\frac{1}{8}$ flip 3 bits, or with probability $\frac{1}{16}$ flip 4 bits, and so on ... . This mutation probability distribution can easily be implemented by sequentially flipping a fair coin: if heads comes up we mutate 1 bit and stop, otherwise we flip the coin again, and if heads is up we mutate two bits and stop. If the coin did not came up heads we continue flipping it. After the number of bits to be mutated is computed, we randomly select and flip them.

3 Experiments

1. Experiment 1 uses 2-point crossover as variation operator (no mutation) on 6 fitness functions: Uniformly Scaled Counting Ones, Linearly Scaled Counting Ones, Tightly Linked Deceptive Trap, Randomly Linked Deceptive Trap, Tightly Linked Non-Deceptive Trap, and Randomly Linked Non-Deceptive Trap.

2. Experiment 2 uses uniform crossover as variation operator (no mutation) on 4 fitness functions: Uniformly Scaled Counting Ones, Linearly Scaled Counting Ones, Deceptive Trap, and Non-Deceptive Trap (for uniform crossover the linkage has no impact).

3. Experiment 3 uses uniform crossover and mutation as variation operators on 4 fitness functions: Uniformly Scaled Counting Ones, Linearly Scaled Counting Ones, Deceptive Trap, and Non-Deceptive Trap.

The population size N is an important parameter for genetic algorithms. We only consider multiples of 10 and use bisection search to find the minimal required population size. Start with $N = 10$ and if successful you are done. If not double the population size ($N = 20$) and try again. Keep doubling until the optimal solution is reliably found or until

the population has reached a maximum of $N = 1280$. When a population size is found that is successful, bisection search is applied to find the smallest population necessary. For instance, say $N = 250$ is the minimal population size needed to solve a problem reliably, then the bisection search would try the following sequence of population sizes: $N = 10, 20, 40, 80, 160, 320, 240, 280, 260, 250$.

GAs are stochastic algorithms so we perform multiple runs to average out the stochastic effects. A problem is solved reliably when 29 out of 30 independent runs find the optimal solution. When no offspring solution has entered the next generation, then the GA run is stopped.

4 deliverables

You need to email **two separate** files:

1. A report in .pdf form
2. A tar archive file of the source code

The report should describe the following things:

- what exactly did you do
- what did you observe
- what conclusions can you draw from your results

If an experiment is successful you need to:

1. plot the number of successes (out of 30 runs) for each population size tried (pop size on the X-axis)
2. report the number of function evaluations and the corresponding CPU time required to generate the global optimum when using the minimal required population size

Practical Assignment 2

5 Linkage Tree Genetic Algorithm

The Linkage Tree GA (LTGA) is a model-based EA that is capable of learning dependencies between bit values in the population, and use these dependencies to perform problem adjusted crossover. Implement LTGA and test it on the same fitness functions as above: Uniformly Scaled Counting Ones, Linearly Scaled Counting Ones, Randomly Linked Deceptive Trap, and Randomly Linked Non-Deceptive Trap (use the randomly linked versions to avoid any possible bias). The deliverables are also the same as in the first part.

C code of LTGA is available at http://homepages.cwi.nl/ ~bosman/source_code.php . You can use this code to check for detailed information but you should still implement your own LTGA (in any language you want except C). Note that the LTGA code uses some mechanisms we did not discuss during the lectures, namely forced improvement and the reciprocal nearest neighbor chain clustering algorithm. You do not need to implement the forced improvement and you can use any hierarchical clustering algorithm you want (there are many available on the web).