Practical Assignment Evolutionary Computation

1 Genetic Algorithm

In the first part 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 and have a stringlength $\ell = 100$. 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.

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 specific genetic algorithm (GA) we will use in the assignment is a so called incremental (or steady-state) GA: this means that we do not work with separate generations but that we generate a solution pair and immediately add one or both solutions to the current population if the fitness score of the solution is better than the lowest scoring solution in the current population:

1. Select and copy two parents from the population with tournament selection (tournament size s).
2. With probability p_c recombine the copies of the parents, with probability $1-p_c$ mutate both copies. When doing mutation first decide how many bits will be mutated: with probability $\frac{1}{2}$ flip 1 bit, with probability $\frac{1}{4}$ flip 2 bits, with probability $\frac{1}{8}$ flip 3 bits, ... until a mutation actually took place. This mutation probability distribution can easily be implemented by sequentially flipping a fair coin.

3. Let the best offspring solution compete with the current worst solution in the population: if the fitness of the new solution is better or equal then it replaces the worst solution in the population.

Run experiments with different parameter values:

- Compare different population sizes (only consider multiples of 10).
- Compare different crossover/mutation probabilities ($p_c = 1, \frac{1}{2}, 0$).
- Compare different tournament sizes ($s = 1$ and 2) (note that $s = 1$ means just a random selection).
- Compare two different crossover operators: uniform crossover versus 2-point crossover. For 2-point crossover, you should do the experiments with tightly linked and randomly linked Trap Functions. Tightly linked means that the 25 subfunctions are placed adjacent to each other on the binary string. With random linkage the 100 bits are placed at a random position on the binary string.

Running an experiment with a particular parameter setting requires executing 50 independent runs. The stopping criterium is met when the global optimum is reached or when a maximum number of solutions have been generated (choose a sensible maximum number). Count the number of generated solutions until the first occurrence of the global optimum (= 'the first hitting time'). A setting is successful if 49 out of the 50 runs reach the global optimum.

2 MLS, ILS, and GLS for Graph Bipartitioning

The goal of the second part of the practical assignment is to implement and experimentally compare multi-start local search (MLS), iterated local search (ILS), and genetic local search (GLS) for a graph-bipartitioning (GP) problem.

Local search algorithms iteratively change a solution until no better solution is found in the neighborhood of the current solution. The local search algorithm we use is the vertex swap neighborhood search (VSN). MLS, ILS, and GLS are metaheuristic algorithms that improve the performance of the local search algorithm.

1. *Multi-start local search* simply restarts local search from a set of randomly generated initial solutions. The best local optimum found is returned as final solution.
2. *Iterated local search* mutates the current solution found by local search and applies local search from the mutated solution. When the new local optimum is better than the current one, ILS continues its search from this new local optimum, else it simply returns to the previous local optimum. The size of the mutation is a crucial factor in ILS. If the mutation is too small, the local search will return to the local optimum it has just mutated. If the mutation is too large there will be no correlation between the new local optimum and the one that was mutated. ILS is then reduced to a MLS algorithm.
3. *Genetic local search* keeps a population of the best solutions found and generates new starting solutions for local search by recombining two randomly selected solutions from the population. If the new local optimum is better than the worst of the population it replaces this solution unless there is already an exact copy present in the population.

3 Experimental study

Set up an experimental study that investigates the following algorithms. Note that to experimentally compare two algorithms you need to run the algorithms for at least 30 times each and compare the mean or median results of these 30 runs with each other. Use a statistical significance test (t-test or Wilcoxon-Mann-Whitney test) to check whether the observed differences are indeed statistically significant.

1. Implement MLS with the VSN local search. Generate 1000 local optima for a single run of MLS. Measure the computational time required in real clock time or in some basic algorithmic step like the number of vertex moves.
2. Specify 3 different mutation sizes and run ILS with the VSN local search for each mutation value. Are the three ILS versions statistically better/worse than MLS ? Are the results obtained with the three mutation sizes statistically different from each other ?
3. Run 2 versions of GLS (using the VSN local search), one with population size 25, the other with size 50. Are the results obtained with the two GLS versions statistically different ? Are the two GLS versions statistically better/worse than MLS or ILS ?
4. Select a topic to investigate. Clearly specify the research goal and discuss the results obtained. Possible topics might be the influence of selection pressure and/or the population size in GLS, the addition of mutation in GLS, an adaptive mutation step size in ILS, comparison with other metaheuristics like simulated annealing or tabu search, ...

   Compare the above algorithms in two ways: first, compare them on the basis of generating the same number of local optima (= 1000); second, compare them on the basis of the same run time (= real clock time or in some basic algorithmic steps like the number of vertex moves).

4 Graph Bipartitioning

- The experiments need to be run on two graphs of size 500: graph-U and graph-G.
- Solutions to the GP problem are represented by binary strings $x_1 \dots x_\ell$ with ℓ the number of nodes in the graph and $x_i \in \{0, 1\}$ specifying to which of the two partitions the node i belongs. Note that the number of ones and zeroes in each solution string needs to be equal to $\ell/2$.
- Mutation in ILS is done by swapping a certain number of nodes. A vertex swap is simply replacing a 1 by 0, and a 0 by 1 in the binary string.
- Recombination is done by uniform crossover that respects the equality constraint between the number of ones and zeroes:
  1. Compute the hamming distance between the two parents. If the distance is larger than $\ell/2$ invert all bit values of one parent.
  2. Generates a single child that has the same bit value as the parents whenever the two parents have the same value for that vertex.
  3. The positions in the child string where the two parents disagree are randomly filled in under the constraint that the total number of ones and zeroes in each solution needs to be equal to $\ell/2$.

5 Report

Write a report discussing your results and send it to dirk@cs.uu.nl. The report should be in **PDF format !!!** The source code should be in SEPARATE, compressed archive file (program.tar.gz). **Do NOT include** your report.pdf file in this archive !

6 Deadline

Part 1: Sunday, **January 6**, 2013, 24:00 hrs.
Part 2: Sunday, **February 3**, 2013, 24:00 hrs.

7 Questions

Any remaining questions about the assignment can be asked during the break or after each lecture.