Assignment 5

Evolutionary Computation

Deadline March 10, 2024

Handout for the *Natural Computing* lecture, February 22, 2024

Johannes Textor, Inge Wortel

In this assignment, you are going to test your understanding of evolutionary algorithms, implement some yourself, and investigate the influence of design choices like survivor selection and integration with local search.

Objectives of This Exercise

1. Explain how choices (e.g. fitness, survivor selection) affect selection pressure in a genetic algorithm
2. Implement simple evolutionary algorithms
3. Analyze and visualize the performance of an evolutionary algorithm
4. Design a fair experiment comparing an expanded evolutionary algorithm to a baseline

Exercises

Exercise 5.1 Fitness function and selection pressure (1 point)

Consider the following fitness functions:

(a) $f_1(x) = |x|$
(b) $f_2(x) = x^2$
(c) $f_3(x) = 2x^2$
(d) $f_4(x) = x^2 + 20$

1. Given each fitness function above, calculate the probability of selecting the individuals $x = 2$, $x = 3$, and $x = 4$ using fitness-proportional selection.
   - provide a table with fitness function value and selection probability for each individual and each function.
   - plot the pie chart ("wheel of fortune") with selection probability of the three individuals for each function.
2. What can you conclude about the effects of fitness scaling on selection pressure?

Exercise 5.2 Role of selection in GAs (2 points)

In this exercise, you will perform a small experiment to examine the role of selection pressure in a simple genetic algorithm (GA). Consider the following (1+1)-GA for binary problems:

- Step 1: Randomly generate a bit sequence x
- Step 2: Create a copy of x and invert each bit with probability μ. Let x_m be the result.
- Step 3: If x_m is closer to the goal sequence than x, replace x with x_m.
- Step 4: Repeat steps 2 and 3 until the goal sequence is reached.

The Counting Ones problem amounts to fit a bit string whose sum of entries is maximum.

1. Implement a simple (1+1)-GA for solving the Counting Ones problem. Use bit strings of length $l = 100$ and a mutation rate $\mu = 1/l$. For a run of 1500 generations, plot the best fitness against the elapsed number of generations. Does the algorithm find the optimum?
2. Now replace Step 3 in the above algorithm with the following: "replace x with x_m". Again run 1500 generations, and plot the result together with those of the original algorithm. What do you see?

3. Can you conclude anything on the difference between these two versions based on the above results? If yes, why? If not, what should you do to make a fair comparison? (Hint: the results of a GA are stochastic since steps like parent selection, mutation, and crossover all involve stochastic operations that can differ from run to run. What does that imply?)
4. Using your insights from the previous question, create a figure that demonstrates the performance difference (if any) between the two versions of the algorithm. Is there a difference in performance? Justify your answer.

Exercise 5.3 Exploitation versus exploration and population diversity (4 points)

Implement a string search genetic algorithm like the one in the lecture, but now with:

- tournament selection with a tunable parameter K,
- your own target string of length L of (approximately) 15 characters,
- an alphabet Σ containing all 26 lowercase letters and all 26 capital letters,
- crossover with probability $p_c = 1$,
- a tunable mutation rate μ,
- a population size $N = 200$
- a fitness as defined in the lecture
- generational replacement with no elitism. Note: since you are doing crossover between two parents, you will now generate two new children at once, so to get a new population of N strings you need to repeat this $N/2$ times.

1. Using $K = 2$ and $\mu = 1/L$, run the algorithm 10 times to measure the time t_{finish} (in generations) needed to find the target.
2. Repeat the experiment at both $\mu = 0$ and $\mu = 3/L$. Do you find the target? Explain your observations. (Hint: you may want to stop trying after some predefined number G_{max} of generations – I suggest 100). Visualize the distributions of t_{finish} for each μ (I would suggest a so-called beeswarm plot).

To evaluate the performance of your algorithm, it is useful to not look only at fitness, but also at population diversity. A lower diversity may indicate that the algorithm is converging, but it also means that we are zooming in on a local optimum and cannot reach the target anymore. Have a look at the Appendix at the end of this document for some ways to analyze population diversity.

3. Rerun 1-2 above, but now also analyze population diversity (e.g. mean pairwise Hamming distance) throughout the run, every 10 generations.
4. (Optional) : plotting a sequence logo of the population in the last generation may also be informative.
5. What do you conclude about the influence of μ on algorithm performance and population diversity?

Optional bonus questions (if your implementation works, this should not be too much extra work, but it is not obligatory for the assignment):

6. Starting from $\mu = 0$, gradually increase μ in steps (choose a range of about 10 values that makes sense to you). Plot the relationship between μ and t_{finish}. What optimal μ do you find?
7. Now set $K = 5$ and repeat the above experiment. What do you find? Explain your observations.

Exercise 5.4 Memetic algorithms (3 points)

Recall the simple EA for the TSP described in our first lecture (see slides). Implement both this algorithm and a variation based on memetic algorithms (MAs). Use the 2-opt algorithm as a local search technique in the memetic algorithm. The 2-opt algorithm tries to swap all pairs of cities to see if this improves the length of the tour (see, e.g. `https://en.wikipedia.org/wiki/2-opt`).

You will investigate the effect of adding local search on performance. Please do so on two datasets:

- the TSP problem instance given in the file 'file-tsp'. This file contains a 50 x 2 matrix with coordinates (x_i, y_i) for each city $i = 1, ..., 50$
- a small instance of your choice selected from the "Symmetric Traveling Salesman Problem" benchmark instances available at `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

1. Compare your MA to the simple EA from the lectures by running both algorithms for for 1500 generations. Please repeat each experiment 10 times to account for stochasticity, and perform this comparison for both datasets, asking: does addition of local search improve performance?
2. Do you think the simple EA with the same number of generations is a fair baseline to answer this question? Why/why not? If your answer is no, then please explain how you would perform an experiment with a fair comparison.

Product

Hand in your assignment in pdf format containing the following sections:

- Since this assignment consists of several smaller exercises, you do **not** need to write an overarching introduction or problem statement this time;
- But as always, it should be clear from your written assignment how you generated each result; please write these details in a short "Methods" section (you can use subsections for each individual exercise). Details like chosen target string, exact parameters and other implementation choices should all be mentioned here.
- In you Results section, please make one subsection for each exercise. Make sure to include the answers to each of the questions for that exercise, along with any requested figures or other outputs.
- In your conclusion section, outline what "take-home message" you derived from each exercise. What did you learn about the factors determining success in evolutionary algorithms?
- Please also provide your code (as an appendix or using a URL to a Github repository), along with clear instructions on how to reproduce your results using that code.

Hand in your report in pdf format on Brightspace; deadline: March 10th, 2024.

Appendix: Analyzing diversity

Here are some ways to analyze diversity:

- You can evaluate a distance (e.g. Hamming distance) between pairs of strings in your population. If your population is very large, you may want to subsample first so that this does not take too long. You can then look at the mean distance or simply visualize the distribution in a histogram or violin plot.
- If you want to analyze your solutions in more detail, you can compute a position-wise diversity measure by assessing the Shannon entropy for each position i in the string:

  $$H_i = -\sum_{j \in \Sigma} f_{ij} \log f_{ij}$$

  where the sum runs over all letters j in our alphabet Σ, and f_{ij} is the fraction of the population that contains letter j at position i. You could also visualize your population in e.g. a sequence logo (`https://en.wikipedia.org/wiki/Sequence_logo`) based on the Shannon entropy (there are packages in both R and python to do this).