Ministry of Science and Higher Education of the Russian

Federation National Research University ITMO

Evolutionary Computation

Spring

2024

Laboratory work No. 3

GENETIC ALGORITHM FOR THE PROBLEM OF OPTIMIZATION OF A CONTINUOUS FUNCTION

Steps of completing the implementation of the project:

1- Implementing MyFactory: this class was responsible for generating random values within the range [-5, +5] through the equation: -5.0 + (10.0 * random.nextDouble()), which will serve as the candidate solutions for the algorithm.

2- Implementing MyMutation: in this class, at the beginning, we implemented the standard mutation that mutates every gene in the individual (where a random value is generated for each gene in every individual within the population, and this random value is added to the current value of the gene), this type did not give good results, and therefore, we implemented <u>the uniform mutation</u> operator by adding another parameter called myMutationProb, which is the probability of mutating each individual in the population, where it iterates through each individual in the population and randomly mutates each gene with a specified probability (random.nextDouble() < mutationProbability). While implementing mutation process, we made sure that the implementation respects the problem constraints, which is maintaining gene values within the defined domain [-5, 5] (individual[i] = -5.0 + 10.0 * random.nextDouble()). Ultimately, this class plays a vital role in enhancing the algorithm's effectiveness in finding optimal solutions.

3- Implementing MyCrossover: in this class, first, we tried the single-point crossover, which selects a random index from the individual, where new genes that have less index will be taken from the first parent, and the rest from the second parent. The single-point crossover gave bad results, and therefore, we implemented <u>the arithmetic crossover</u> operator. This operator takes two parent solutions (`p1` and `p2`) and generates a single child solution (child[j] = alpha * p1[j] + (1 - alpha) * p2[j]). The crossover is controlled by a parameter `alpha`, randomly generated within [0, 1], which determines the contribution of each parent to the child solution. For each gene, the child value is computed using a weighted sum of the corresponding parent values. This crossover ensures exploration of the solution space by combining features from both parents while maintaining consistency with problem constraints.

4- Changing selection strategy: the selection strategy that was used at the beginning is the roulette wheel, which selects randomly from individuals with respect to their probability that is proportional with the individual fitness. This strategy is raising the chance of selecting the good individuals but still not guaranteeing that, which resulted in a good but not excellent performance for the algorithm. Therefore, the selection strategy was replaced with the tournament selection strategy with a probability of 0.9, which will give a better chance to choose better parents for the next generation.

After implementing the code, with changing the values of parameters, many experiments were conducted, where the best results were given as following.

Problem Size	Population Size	Number of Iterations	Mutation Probability	Tournament selection probability	Result
2	8	100	0.05	0.9	9.73
10	10	300	0.03	0.9	9.74
20	20	700	0.02	0.9	9.82
50	50	1500	0.01	0.9	9.71
100	100	5000	0.01	0.9	9.55

Questions:

1. What is more important, crossover or mutation?

- In general, the mix of those both two operators is often giving the best results, but this can be ensured for all situations. Besides, we can't totally decide which operator is more important because it can depend on the context of the problem.
- But we can say that in our situation, where the objective function is The Ackley function which is known for having multiple local optima distributed across the search space, and therefore, crossover alone is not sufficient to escape. In such cases, mutation can provide the necessary exploration to discover new regions of the search space. And this can be proven after conducting experiments on the code, where some experiments involved using only mutation operator, and some involved only crossover. Average of fitness values that resulted from mutation operator where about 8.2, while when using only crossover operator, it was about 7.5.
- But as we said, the best results can be obtained from using both operators together, where results are reaching values higher than 9.5.

2. How does the value of the "population size" parameter affect the performance and efficiency of the algorithm?

Population size parameter has big affect on the algorithm, where it's playing a big role in the following points.

A) On the performance side:

- Exploration vs. Exploitation: Larger populations enable more exploration of the search space, while smaller populations may lead to quicker convergence.
- Convergence Speed: Larger populations generally converge more slowly but may find better solutions, while smaller populations converge faster but risk premature convergence.
- Robustness: Larger populations are more robust to noise but require more resources, while smaller populations are more susceptible to getting stuck in local optima.

B) On the efficiency side:

- Computational Cost: Larger populations require more resources for evaluation and operations, while smaller populations are computationally cheaper.
- Parallelization: Larger populations benefit more from parallelization, while smaller populations may not utilize parallel resources effectively.

And therefore, the population size should be chosen based on the problem's characteristics and available computational resources, balancing exploration, convergence speed, and computational cost.

3. Is it important to know the domain of definition of the variables of the objective function?

Yes of course, because as we saw in our example, understanding the domain helps in designing appropriate representation schemes for all sides (candidate solutions, defining suitable initialization strategies, and ensuring that operators like crossover and mutation produce valid offspring within the specified range). It will help avoiding meaningless explorations, and therefore, directing the algorithm towards the optimal solution faster, and avoiding wasting resources and time, which helps in increasing efficiency and performance.