ARTIFICIAL INTELLIGENCE

2nd semester

WORKSHEET – GENETIC ALGORITHMS

Genetic algorithms are parallel and stochastic search algorithms, inspired in the Darwin's natural selection theory and molecular biology. The search process is said to be parallel because a set of *n* solutions is kept at each moment and not just one as, for example, in the Hill-climbing algorithm. In the genetic algorithms terminology, this set is called *population*. Figure 1 shows the general structure of a genetic algorithm.

```
t = 0
create_initial_population(P(t))
evaluate(P(t))
while stop condition not met
        P'(t) = select(P(t))
        P''(t) = apply_genetic_operators(P'(t))
        P(t + 1) = create_next_population(P(t), P''(t)))
        evaluate(P(t + 1))
        t = t + 1
return best individual found
```

Figure 1 – General structure of a genetic algorithm.

In general, the initial *population* is built randomly. However, domain knowledge can be used.

Each *individual* (solution) from the population is evaluated using a *fitness function*. The returned value should reflect the quality of the individual as a solution to the problem.

The search process progresses during some number of *generations* (iterations). At each generation, the first step consists in selecting *n* individuals. The *selection method* should obey to the following principles: 1) The best individuals have more chances of being selected and 2) selection is done with reposition (that is, each individual may be selected more than once). In this worksheet we will use two selection methods: *roulette wheel* and *tournament*. The selection method outputs an intermediary population *P'(t)*, where *t* represents the generation number.

The next step consists in the application of genetic operators to the individuals of *P'(t)*. These operators are applied in order to create new and different individuals, hopefully, better than the existing individuals, allowing the algorithm to explore other regions of the search space. The most common operators are the *recombination* (also known as *crossover*) and *mutation* operators. The first one consists in creating two new individuals by mixing genetic material from two existing individuals. The mutation operator generates a new individual by modifying the genes of one existing individual. Typically, the mutation operator is applied after the recombination operator. The application of both operators is stochastic in nature. From the application of the genetic operators, a new intermediary population *P''(t)* is generated.

Population *P(t+1)* can be built using individuals from both population *P(t)* and population *P''(t)*. However, the most common method, named *generational strategy*, consists in doing *P(t+1)* = *P''(t)*. The last step in each generation consists in evaluating the new population.

In the end of the search process, the algorithm returns the best individual found, hoping that it is a good solution to the problem.

Exercise 1

The *Knapsack* problem is a classic search optimization problem that can be described as follows:

Let us consider that we have *N* items. Each item *i* is characterized by its weight w_i and its value v_i. There is also a sack where items can be put. The weight of the items put in the sack cannot exceed some value *W*. The goal is to choose a subset of the *N* items to be put in the sack so that the total value of the items is maximized while the total weight does not exceed *W*.

In this exercise you should implement a genetic algorithm that can be used to find a good solution to this problem. Therefore, you should complete the project provided with this worksheet by implementing the following methods:

a) **`run()`** from class **`GeneticAlgorithm`**.

b) **`tournament()`** from class **`Tournament`**.

c) **`recombine()`** from class **`RecombinationUniform`**.

d) **`mutate()`** from class **`MutationBinary`**.

e) **`compute_fitness()`** from class **`KnapsackIndividual`**. Please, notice that an individual can be invalid (when the total weight of the objects exceeds the value of *P*). What should we do in these situations? There are some alternatives that will be discussed in the class.

Exercise 2

You should perform tests that allow to study the following aspects:

- The effect of varying the population size;
- The effect of varying the tournament size;
- The effect of varying the probabilities of the genetic operators;
- The performance of the algorithm when the roulette wheel selection method versus the performance of the algorithm when the tournament selection method is used;
- The performance of the algorithm when the one cut, two cuts and uniform recombination operators are used.

You should perform at least 30 independent experiments with each algorithm configuration so that the obtained results have some statistical meaning.

Exercise 3

The travelling salesman problem is a classic graph traversing problem that consists in discovering the shortest trajectory that allows a salesman to visit a set of cities exactly once and return to the departure city. Formally, we can define the problem as follows: given a set of points and the distances between each pair of points, find the minimum cost trajectory that allows to visit each point exactly once.

Please, implement a genetic algorithm to solve the travelling salesman problem.