Constraint Handling

In an optimization problem, we can identify some key concepts:

- **Objective function**: it is a numeric value that represents how good is a solution. It could be money, time, energy, error, force, space, etcetera.
- **Variables**: are the values that we can change for improving the objective function.
- **Constraints**: Sometimes, features' values must accomplish some requirements. For example, the number of people working in a process needs to be more or equal to 1.

The presence of constraints implies that not all possible combinations of variable values represent valid solutions to the problem at hand. Unfortunately, constraint handling is not straightforward in an EA, because the variation operators (mutation and recombination) are typically "blind" to constraints. That is, there is no guarantee that even if the parents satisfy some constraints, the offspring will satisfy them as well.

		Objective function	
		Yes	No
Constraints	Yes	**Constrained optimization problem** Knapsack problem	**Constraint satisfaction problems** 8 queen problem Sudoku
	No	**Free optimization problem** Linear Regression Image Transformation	**There is no problem**

Constraint Handling can be performed:

1. Indirect constraint handling (transformation) coincides with penalizing constraint violations.
2. Direct constraint handling (enforcement) can be carried out in two different ways:
    a) Allowing the generation of candidate solutions that violate constraints: repairing infeasible candidates.
    b) Not allowing the generation of candidate solutions that violate constraints: preserving feasibility by suitable operators (and initialization).
3. Mapping constraint handling is the same as decoding, i.e., transforming the search space into another one.

In general, the presence of constraints will divide the space of potential solutions into two or more disjoint regions, the **feasible region** (or regions), containing those candidate solutions that satisfy the given feasibility condition, and the **infeasible region** containing those that do not.

PENALTY FUNCTIONS

Assuming a minimization problem, the use of penalty functions constitutes a mapping from the objective function $f(x)$ to $f'(x)$ such that

$$f'(x) = f(x) + P(d(x,F))$$

where F is the feasible region as before, $d(x,F)$ is a distance metric of the infeasible point to the feasible region (this might be simply a count of the number of constraints violated) and the penalty function P is monotonically increasing nonnegatively such that $P(0) = 0$.

If the penalty function is too severe, then infeasible points near the constraint boundary will be discarded, which may delay, or even prevent, exploration of this region. Equally, if the penalty function is not sufficient in magnitude, then solutions in infeasible regions may dominate those in feasible regions, leading to the algorithm spending too much time in the infeasible regions and possibly stagnating there. In general, for a system with m constraints, the form of the penalty function is a weighted sum

$$P\big(d(x,F)\big) = \sum_{i=1}^{m} w_i d_i(x)$$

The function $d_i(x)$ represents the distance from the point x to the boundary for the constraint i, and w_i represents the weight of the constraint i.Penalization functions are classified as static and dynamic.

Static Penalty Functions: Three methods have commonly been used with static penalty functions, namely *extinctive* penalties (where all of the w_i are set so high as to prevent the use of infeasible solutions), *binary* penalties (where the value d_i is 1 if the constraint is violated, and zero otherwise), and distance-based penalties. It has been reported that of these three the latter gives the best results, and the literature contains many examples of this approach. This approach relies on the ability to specify a distance metric that accurately reflects the difficulty of repairing the solution, which is obviously problem-dependent, and may also vary from constraint to constraint. However, the main problem in using static penalty functions remains the setting of the values of w_i. In some situations, it may be possible to find these by experimentation, using repeated runs and incorporating domain-specific knowledge, but this is a time-consuming process that is not always possible.

Dynamic Penalty Functions: One approach is described as follows. A population is evolved in a number of stages - the same number as there are constraints. In each stage i, the fitness function used to evaluate the population is a combination of the distance function for constraint i with a death penalty for all solutions violating constraints $j \neq i$. In the final stage all constraints are active, and the objective function is used as the fitness function. It should be noted that different results may be obtained, depending on the order in which the constraints are dealt with.

REPAIR FUNCTIONS

The repair algorithm works by taking an infeasible point and generating a feasible solution based on it. In some problems, this technique is relatively simple. However, in general, defining a repair function may be as complex as solving the problem itself.

In continuous optimization problems, the most popular methodology consists of taking an infeasible point and another feasible point and using binary search for finding a feasible point that stands in the segment joining those points.