All evolutionary algorithms use ideas that resemble the process of evolution. There are four broad categories: A Genetic Algorithm (GA) uses bit-string representations of genes, while Genetic Programming (GP) uses binary codes to represent programs or instruction sets. Evolutionary Strategy (ES) require the use of vectors. Lastly, Evolutionary Programming (EP), which closely resembles ES, without imposing a specific data structure type[1, 2]. Our approach falls into the category of GA. We refer to the literature review of Vikhar for more insights into the field. The most vital concept in this category is the *gene* representation.

For the algorithm, we follow a rigid structure of the operations as outlined in 1. As 1 shows, we define 5 fundamental operations. Initiation, Selection, Crossover, Mutation and Recombination. The core idea is to generate new individuals every generation while discarding those who are not deemed fit enough for the next iteration cycle. This optimization method differs from gradient-based methods such as Deep Learning, because it does not require us to use differentiable functions. This makes evolutionary algorithms tremendously useful but also highly dependent on the composition of the fitness function.

Algorithm 1 The basic structure of an evolutionary algorithm.

```
Require: Hyperparameters
Ensure: The result is the final population
  $population \leftarrow$ INIT $population$;
  while not $termination$ do
      $parents \leftarrow$ SELECT $population$;
      $offspring \leftarrow$ CROSSOVER $parents$;
      $mutants \leftarrow$ MUTATE $offspring$;
      $survivors \leftarrow$ RECOMBINE $population \cup mutants$;
      $termination \leftarrow$ DETERMINE $termination$
      $population \leftarrow survivors$
  end while
```

The initiation operator refers to the creation of the initial set of candidates for the selection process in the first iteration of the algorithm. Often, this amounts to the random generation of individuals. However, choosing among a subset of the search space can allow for a faster convergence.

The selection operator chooses a set of individuals among the population according to a selection procedure. These individuals go on to act as material to generate new individuals. This operator can strongly influence the level with which the algorithm explores the search space. For instance, if an algorithm only selects the best individuals it is easy to get stuck within

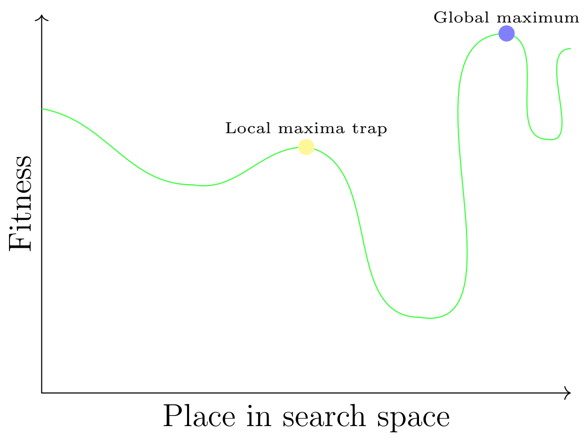

Figure 1: Schematic example showing various local optima.

an local maximum. Figure 1 displays why local maxima should be avoided. Mainly, if we get stuck, we may not be able to find the absolute best solution.

Within the crossover procedure, we select random pairing of individuals to pass on their characteristics. We can often generate at least two additional offsprings, if we have two parents and just reverse the operation.

Mutations introduce random pertubations to the offsprings. This extends the search space beyond what is available by the parents.

The Recombination operation decides which individuals remain in the population for the next iteration[1]. This operator-type is a second source that determines the exploration space of the evolutionary algorithm. For instance, if we only allow the best 10 individuals to move on to the next iteration and only select the top 3 individuals for the crossover phase, we quickly converge towards one solution. Hence, both operators interact, which is why the literature often treat these operations as identical. However, splitting them allows us to control the number of sampled off spring and the population size separately.

We name the strict selection of the best individuals among the offsprings and the previous population *Fittest-Survivor-Recombination*. This recombiner strictly optimizes the population and is susceptible to getting stuck in local minima. In contrast, we name the addition of the top-k best offsprings to the initial population *Best-of-Breed-Recombination*. The former will guarantee, that the population size remains the same across all iterations but is prone to local optima. Furthermore, we propose one additional recombination operator. The operator selects the new population in a different way than the former recombination operators. Instead of using the viability directly, we sort each individual by every viability component, separately. This approach allows us to select individuals regardless of the scales of every individual viability measure. We refer to this method as *Ranked-Recombination*.

[1]We have to point out that in the literature, recombination is often synonymous with crossover. Both steps are similar in their filtering purpose. However, the selector filters potential parents while the recombiner filters the population. However, in this thesis recombination refers to the update process which generates the next population.