5 Genetic/Evolutionary Algorithms

Evolutionary algorithms (EA) is a meta-heuristic that uses observations from the theory of evolution to solve optimization problems of various types. In general, evolutionary algorithms try to optimize an *objective function* F (also called *fitting function*) by *reproduction* and *selection of a population* (a. k. a. swarm) of *individuals*, i. e., of solution candidates for the optimization problem to be solved, improved iteratively in *generations*.

Reproduction involves

- *inheritance* (traits (characteristics) from the "parents" are passed on to the "children") and
- *variation* and *mutation* (minor random modification of the properties of the "children"), resp.

The latter is necessary to allow a better (complete) exploration of the search space.

The **selection** is based on an *evaluation* of the individuals according to the objective function to be optimized and ultimately ensures that "better" individuals have a higher chance of reproduction than less good individuals. Furthermore, the selection process can be used to select those individuals that will be carried over into the new generation, i. e., to select the best "parents" and "children." This leads to the fact that the search is steered in the direction of good solution candidates.

A distinction is made between the *genotype* and the *phenotype* of an individual. The term "genotype" refers to the coding of an individual, whereas the term "phenotype" refers to the coded individual itself.

By μ, we denote the population size, i. e., the number of individuals in each generation. We denote the number of new offspring generated in each generation by λ. A corresponding evolutionary algorithm is called (μ, λ)-EA and is presented schematically below.

Algorithmus 7 (μ, λ)-EA

input: F: objective function to be optimized
output: r: best solution candidate found
  $t := 0$
  $P(0) :=$ generate the initial population consisting of μ individuals
  evaluate the individuals in $P(0)$: $\mathrm{Eval}_F(P(0))$
  $r :=$ best individual in $P(0)$
  while termination criterion not met **do**
    $P' :=$ select by a certain selection operator from $P(t)$ parents for λ offspring
    $P'' :=$ create λ offspring from P' using a crossover (a.k.a. recombination) operator
    $P''' :=$ mutate the individuals from P'' using a certain mutation operator
    evaluate the individuals in P''': $\mathrm{Eval}_F(P''')$
    if better individual found than r **then**
      $r :=$ new, best individual found in P'''
    end if
    $P(t+1) :=$ select by a certain selection operator μ individuals from $P(t) \cup P'''$
    $t := t + 1$
  end while

Note: The best individual found, i. e., the solution candidate that has the best value of F, does not necessarily have to be included in the last generation since it may have been removed by selection.

This algorithmic scheme describes the so-called *Evolutionary Cycle* presented in Figure 3.

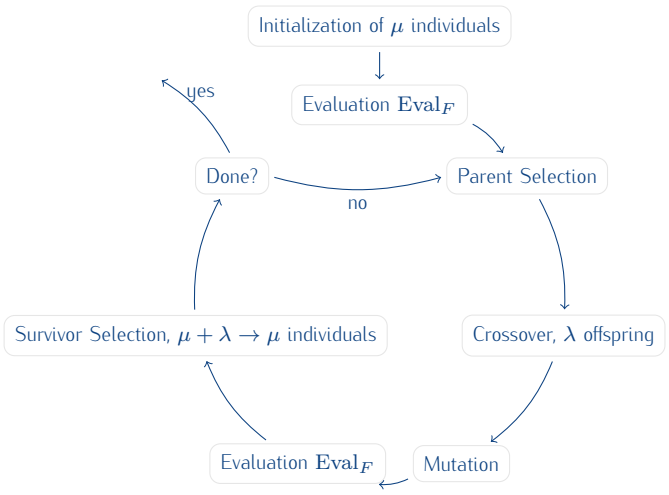
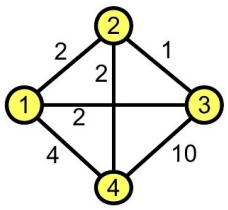


Figure (3) *The Evolutionary Cycle, base of Evolutionary Algorithm* (μ, λ)*-EA*

5.1 An Instructive Example: Traveling Salesperson Problem

Using the *Traveling Salesperson Problem* (TSP), possible selection, crossover, and mutation operators are discussed below.

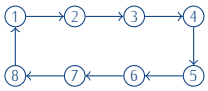
Given a complete, edge-weighted graph G on N nodes, we are looking for a shortest tour that visits each node exactly once. Here we see an example with $N = 4$.



An optimal solution is $(3 \to 2 \to 4 \to 1 \to 3)$ which has length 9.

The phenotype of an individual is an actual tour, the genotype of an individual is the permutation that specifies the visiting order of the nodes. In the example above, the coding is $(3, 2, 4, 1)$.
On the left, we see the genotype coding of the right tour.

1	2	3	4	5	6	7	8



5.1.1 Selection Operators

The motivation for the selection of individuals is, on the one hand, that only the best fitting individuals of the parent generation should be selected for crossover (recombination) (parent selection), on the other hand

that only the best individuals of parents and children should be selected for the next generation (survivor selection). For example, the following two methods can be used as selection operators:

- **Tournament Selection**: Two or more individuals are randomly drawn from the population, with the best individual surviving or being allowed to recombine. The selection of the best individual can also be done probabilistically, i. e., the best individual wins with higher probability.
- **Rank Selection**: First, a ranking list of the N individuals is created with respect to their quality. I_1 denotes the best fitting individual, whereas I_N denotes the least fitting individual. The individual I_k will then be selected with probability

$$\Pr(I_k) = \frac{N-k}{\binom{N}{2}} = \frac{2}{N} \cdot \left(1 - \frac{k-1}{N-1}\right) .$$

For example, for $N = 10$ we have:

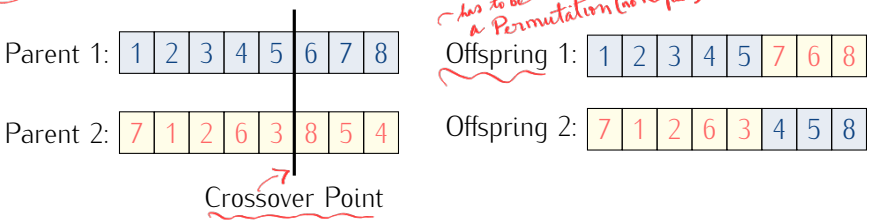
$$\Pr(I_1) = \frac{9}{45},\ \Pr(I_2) = \frac{8}{45},\ \ldots,\ \Pr(I_{N-1}) = \frac{1}{45},\ \Pr(I_N) = \frac{0}{45}$$

In addition, **elitism** can be used. This means that the best individual is automatically carried over to the next generation.

5.1.2 Crossover Operators

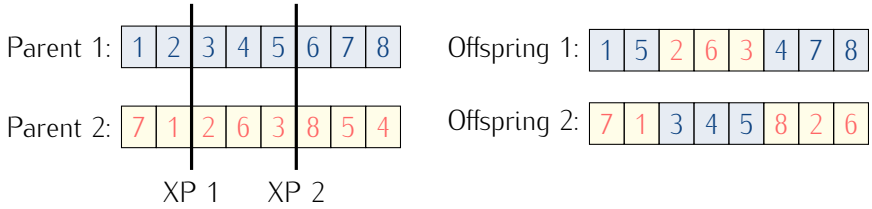
The goal of crossover is that the children inherit as many traits (characteristics) of the parents as possible. The important thing is that this recombination must result in valid individuals. For the *Traveling Salesperson Problem*, this means that the result of the crossover must be a valid permutation, i. e., that the new individual is a feasible round trip. Here, for example, the following recombination operators are suitable:

- **One-point crossover**: A crossover point is chosen at random. For each parent individual, the first part from the beginning through the crossover point is kept, the remaining numbers are given the order they have in the other parent.



So, in the Parent 1 permutation, the sequence $(6, 7, 8)$ is rearranged to $(7, 6, 8)$ as in Parent 2, 7 is to the left of 6, which in turn is to the left of 8.

- **Partially matched crossover (PMX)**: Randomly select two crossover points XP1 and XP2, and interchange the intermediate partial permutation. Now fill up the remaining positions with numbers from the corresponding parent. If this would result in a duplicate number x, consider the partial permutation between the crossover points as a mapping of x to the number in the other parent, and use the assigned number (this may have to be repeated if this number would then also be duplicated). For example, see:



  So, in Offspring 1, the number 2 must not be used on position 2, as it occurs in the intermediate partial permutation. The mentioned mapping is $2 \mapsto 3$. But using 3 is also not allowed on position 2, so one applies the mapping $3 \mapsto 5$, and 5 is feasible at position 2 at last.

- **Edge recombination crossover (ERX)**: The goal of this crossover operator is that as many edges of the offspring as possible have also occurred in a parent.

Algorithmus 8 Edge recombination crossover (ERX)

```
Compute the neighborhood information from the parents (adjacency list)
Start with the first node of a randomly chosen parent
while offspring not completed do
  if there is a neighbor of the node last included then
    Choose as next node the neighboring node with the shortest neighborhood list (random choice to resolve conflicts)
  else
    Choose an arbitrary still unused node
  end if
  Remove the chosen node from all neighborhood lists
end while
```

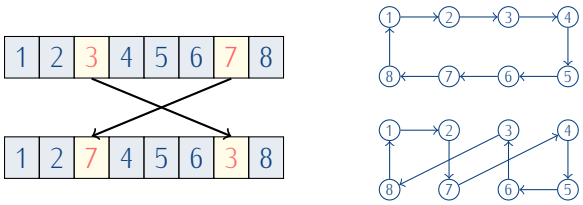
5.1.3 Mutation Operators

The goal of mutation operators is to slightly modify a new individual generated by crossover to allow for better (or even full) exploration of the search space. Again, as with crossover, it is important that the resulting individuals are feasible, i. e., in the case of the *Traveling Salesperson Problem*, they must represent valid round trips.

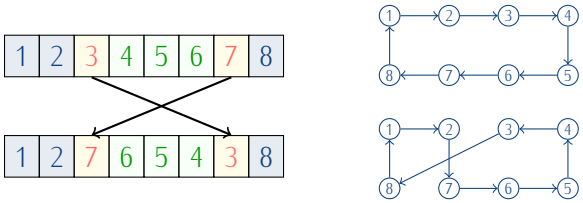
The probability for performing a mutation should be chosen small, in order not to destroy the effects of crossover because the latter tries to combine the positive characteristics of the parents to produce even better offspring.

For the *Traveling Salesperson Problem*, for example, the following mutation operators are conceivable:

- **Swap mutation operator**: Two numbers of the permutation are randomly chosen and then interchanged. In the case of *TSP*, this small modification to the genotype of an individual can cause a significant change in the phenotype, as one can see in the following example where the genotypes can be seen left, and the phenotypes on the right. For example, see:



- **Inversion mutation operator**: Two numbers of the permutation are randomly chosen and then the intermediate sequence (including the chosen numbers) is reversed. Compared to the *Swap mutation operator*, the effects on the phenotype are less substantial here. For example, see:



only 2 connections will change rest will remain same

5.2 Mathematical Analysis of a (1, 1)-EA for Sorting

We now want to present an $(1,1)$ evolutionary algorithm which solves the sorting problem with 1 individual and 1 offspring. Our main goal is to *analyze* the evolutionary algorithm *mathematically*.

Let the input of the algorithm be a permutation (**phenotype**) $\pi \in S_n$, where S_n is the set of all permutations of the numbers from 1 to n. We use the linear sequence $[\pi(1), \ldots, \pi(n)]$ as **genotype** and write $\pi = [\pi(1), \ldots, \pi(n)]$ because everything is clear in this context.

At first, we need an **objective function** that measures the *sortedness* of a permutation. Useful possibilities for such an objective function are:

- $\mathrm{INV}(\pi)$[5]: Number of pairs (i,j), $i < j$ with $\pi(i) < \pi(j)$, i. e., pairs in correct order. π is sorted iff $\mathrm{INV}(\pi) = \binom{n}{2} = \frac{1}{2}n(n-1)$.
- $\mathrm{HAM}(\pi)$[6]: Number of positions with $\pi(i) = i$, i. e., the number of keys already at the final position. π is sorted iff $\mathrm{HAM}(\pi) = n$.
- $\mathrm{LAS}(\pi)$[7]: Length k of a longest ascending subsequence $\pi(i_1) < \cdots < \pi(i_k)$; π is sorted iff $\mathrm{LAS}(\pi) = n$.

For example:

k:	1	2	3	4	5	6	7	8
$\pi(k)$:	3	7	6	8	4	1	2	5

$\mathrm{INV}(\pi) = 11$, $\mathrm{HAM}(\pi) = 0$, $\mathrm{LAS}(\pi) = 3$ (one longest subsequence is marked in red; here, $i_1 = 1$, $i_2 = 5$, and $i_3 = 8$).

The **selection** procedure for the survivor selection is elitism, i. e., either the individual from the previous generation or the individual newly created in this generation by mutation is selected, depending on which of the two individuals has a higher quality with respect to one of the objective functions defined above. There is no parent selection. Subject to the mutation is a copy of the only individual.

The algorithm only executes **mutations**. For this purpose, the following local operations may be used as *part* of a mutation. We call them *local mutation operation*.

- $\mathrm{exch}(i,j)$: The keys at position i and j swap their positions.
- $\mathrm{jump}(i,j)$: The key at position i jumps to position j and all keys in between move one place forward and backward, resp.

Since survivor selection applies elitism and, thus, always only one best individual survives, this algorithm is actually a *hill-climbing* algorithm. To prevent the algorithm from getting stuck in a local optimum, the above operators exch and jump may be used multiple times during mutation, theoretically as often as desired. This guarantees that from each suboptimal permutation π a better permutation π' can be obtained in one mutation step.

[5] INV: Inversions; actually: the non-inversions is counted
[6] HAM: Hamming
[7] LAS: longest ascending subsequence

Therefore, the mutation operator works as follows:

Algorithmus 9 Mutation operator for the sorting EA

input: π: permutation
output: π': mutated permutation
  $\pi' := \pi$
  randomly choose $S \in \mathbb{N}_0$ w.r.t. Poisson distribution[8] with $\lambda = 1$
  {at least one local operation will be executed:}
  for $k := 1$ **to** $S + 1$ **do**
    u.a.r. choose two positions (i, j), $i \neq j$, $i, j \in \{1, \ldots, n\}$
    $\pi' := \begin{cases} \text{with probability } \frac{1}{2}: & \text{apply exch}(i, j) \text{ to } \pi' \\ \text{with probability } \frac{1}{2}: & \text{apply jump}(i, j) \text{ to } \pi' \end{cases}$
  end for

Now, the evolutionary algorithm works as follows:

Algorithmus 10 $(1, 1)$-EA$_{f,n}$ for sorting sequences of length n, $f \in \{\text{INV}, \text{HAM}, \text{LAS}\}$

input: π_0: permutation
output: π: sorted permutation
  $t := 0$
  $\pi := \pi_0$
  while $f(\pi) <$ optimum value **do**
    generate permutation π' by applying the mutation operator to π; $t := t + 1$
    if $f(\pi') > f(\pi)$ **then**
      $\pi := \pi'$
    end if
  end while

We now analyze this evolutionary algorithm in terms of its *expected running time*. We measure the runtime of the algorithm in the expected number of function evaluations of the objective function f. This corresponds to the number of iterations of the while-loop and thus the number of generations generated.

We start with a lower bound, i. e., how many evaluations are *at least* necessary in expectation.

Theorem. The expected number of evaluations of $f(\pi)$ is in $\Omega(n^2)$, for $f \in \{\text{INV}, \text{HAM}, \text{LAS}\}$.

Proof. The probability that the first chosen permutation π_0 is already sorted is $\frac{1}{n!}$.

A new permutation π' generated by the mutation operator is the solution of the sorting problem if *in the last local step* of the mutation the sorted sequence was generated.

For the exch, as well as for the jump-operation there are at most 2 index pairs which turn an unsorted sequence into a sorted sequence. Thus, the probability of actually generating the sorted sequence of keys by this local step is *at most*

$$\left(\frac{1}{2} \cdot \frac{2}{n \cdot (n-1)}\right) \cdot 2 = \frac{2}{n \cdot (n-1)} .$$

[8] $\Pr[S = i] = \frac{e^{-\lambda} \cdot \lambda^i}{i!} \overset{\lambda=1}{=} \frac{1}{e \cdot i!}$. $E[S] = \lambda = 1$, $\text{Var}[S] = \lambda = 1$.

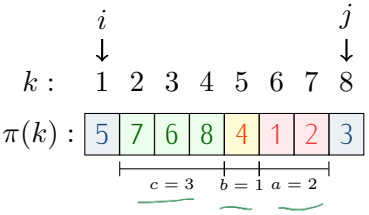
The expected number of evaluations of $f(\pi)$ is equal to the inverse of this success probability, and thus $E[t] \geq \frac{1}{2} \cdot n \cdot (n-1) \in \Omega(n^2)$. □

Theorem. The expected number of evaluations of $f(\pi)$ is in $\mathcal{O}(n^2 \cdot \log n)$, for $f \in \{\mathrm{INV}, \mathrm{HAM}, \mathrm{LAS}\}$.

Proof for $f = \mathrm{INV}$ We consider an index pair (i, j) with $i < j$ and $\pi(i) > \pi(j)$, i. e., a pair of keys which are an inversion in the current permutation π, i. e., a pair in wrong relative order [9]. Between these indices, i. e., at positions $i+1, \ldots, j-1$, there may be further keys, for which there are three possibilities:

- Let a be the number of intermediate keys that are *smaller* than $\pi(j)$
- Let b be the number of intermediate keys that belong *between* $\pi(j)$ and $\pi(i)$
- Let c be the number of intermediate keys that are *greater* than $\pi(i)$

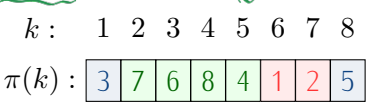
For example, see:



For the two local mutation operations, the value of the objective function changes as follows:

- $\mathrm{exch}(i, j) = \mathrm{exch}(j, i)$
  increases the value of the objective function by $2b + 1$ because the b-keys are now in correct order w.r.t. $\pi(j)$ and $\pi(i)$, so they contribute $2b$. Additionally, $\pi(j)$ and $\pi(i)$ are now in correct order, too, and contribute $+1$.

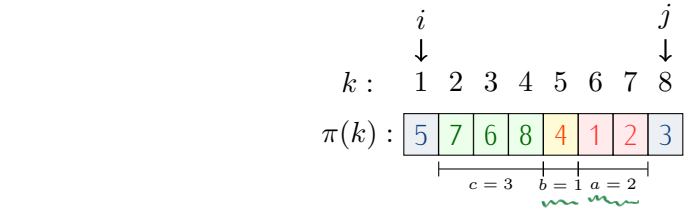
  In our example, we have for $\mathrm{exch}(1, 8)$ an increase of 3:

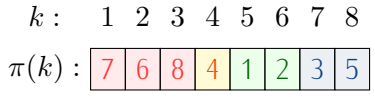


[9] such a pair must exist as the permutation is not yet the sorted sequence

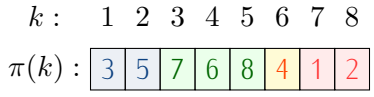
- In order to analyze the effect of the jump operation on the value of the objective function, we need to distinguish which of the chosen positions is i and which one is j:
  - $\text{jump}(i,j)$ (larger key jumps to the right): *changes* the value of the objective function by $a+b-c+1$.
  - $\text{jump}(j,i)$ (less key jumps to the left): *changes* the value of the objective function by $-a+b+c+1$

  As $a, b, c \geq 0$, at least one of the two values is at least 1. That means that with the "right" order of choice of i and j, also the jump-Operation increases the value of the objective function.





(a) *Result of* $\text{jump}(1,8)$



(b) *Result of* $\text{jump}(8,1)$

We only consider the case where exactly one of these local operations is performed by the mutation operator i. e., $S = 0$[10]. The probability of this event is $\Pr[S=0] = \frac{1}{\mathrm{e} \cdot 0!} = \mathrm{e}^{-1}$ (Poisson distribution with $\lambda = 1$).

Each of the two local operations is then executed with probability $\frac{1}{2}$.

Since the exch operation always improves the objective function for an inverse pair, the probability to improve by exch for a given pair simply corresponds to the probability to choose this pair, i. e., to choose appropriate indices i and j:

$$\frac{1}{\#\text{pairs}} = \frac{1}{\binom{n}{2}} = \frac{2}{n \cdot (n-1)}$$

For the jump operation, only (at least) one order of the parameters improves the value of the objective function, i. e., the probability to improve is here at least:

$$\frac{1}{2} \frac{1}{\#\text{pairs}} = \frac{2}{2 \cdot n \cdot (n-1)}$$

In total, the probability to improve the value of the objective function for a given inverse pair by a single operation is at least

$$\mathrm{e}^{-1} \cdot \left(\frac{1}{2} \cdot \frac{2}{n \cdot (n-1)} + \frac{1}{2} \cdot \frac{2}{2 \cdot n \cdot (n-1)} \right) = \frac{3}{2 \cdot n \cdot (n-1) \cdot \mathrm{e}}$$

If there are m inverse pairs, the probability to improve the value of the objective function at least once is at least

$$\frac{3 \cdot (m)}{2 \cdot n \cdot (n-1) \cdot \mathrm{e}}$$

[10] If the other cases are also considered, the probability of producing a sorted permutation *increases*. Since we are looking for an upper bound on the expected running time, this would only improve the result

How often an experiment with probality p for the positive event has to be repeated on average until the event will be observed? These experiments are distributed geometrically, hence the expected numer number of repetitions is $\frac{1}{p}$. So, the inverse of this number, $\frac{2}{3} \cdot \frac{1}{m} \cdot \mathrm{e} \cdot n \cdot (n-1)$ gives us an upper bound on the expected number of generations to improve the value once.

In the worst case, all $\binom{n}{2}$ pairs are in wrong order at the beginning. Hence, the expected value for the total number of generations is at most:

$$E[\#\text{generations for sorting}] = E[t] \leq \sum_{m=1}^{\frac{1}{2}n(n-1)} \left(\frac{2}{3} \cdot \frac{1}{m} \cdot \mathrm{e} \cdot n \cdot (n-1) \right)$$

$$\leq \frac{2}{3} \cdot \mathrm{e} \cdot n^2 \cdot \sum_{m=1}^{\frac{1}{2}n(n-1)} \frac{1}{m}$$

$$\leq \tfrac{2}{3} \cdot \mathrm{e} \cdot n^2 \cdot \left(\ln\left(\tfrac{1}{2}n(n-1)\right) + 1 \right) \in \mathcal{O}(n^2 \log n)$$

Here, we applied the following estimation for the *harmonic series*:

$$\sum_{m=1}^{k} \frac{1}{m} \leq \ln k + 1.$$

□

The objective functions HAM and LAS can be analyzed similarly (→ exercises).

We hope to use Genetic Algo in problem where we dont know the solution / Runtime of our Problem as we know now for Sorting → we know our runtime will be close to specialized problem solver even for unknown problems.