LAB2 - GENETIC ALGORITHM

CONTRIBUTORS

LEONARDO ROLANDI: S301215

FLAVIO PATTI: S301104

INITIAL GENOMES

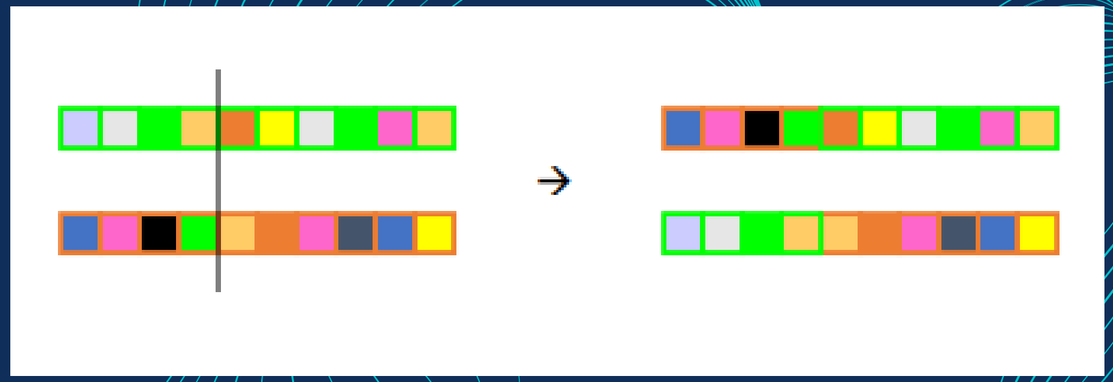
```
population = list()

for genome in [tuple(0 for _ in range(PROBLEM_SIZE)) for _ in range(POPULATION_SIZE)]:
    genome = mutation(genome)
    population.append(Individual(genome, compute_fitness(genome)))
```

The initial genomes of the population are created with all the gene to 0 and then setting randomly only one locus to 1

CROSS OVER

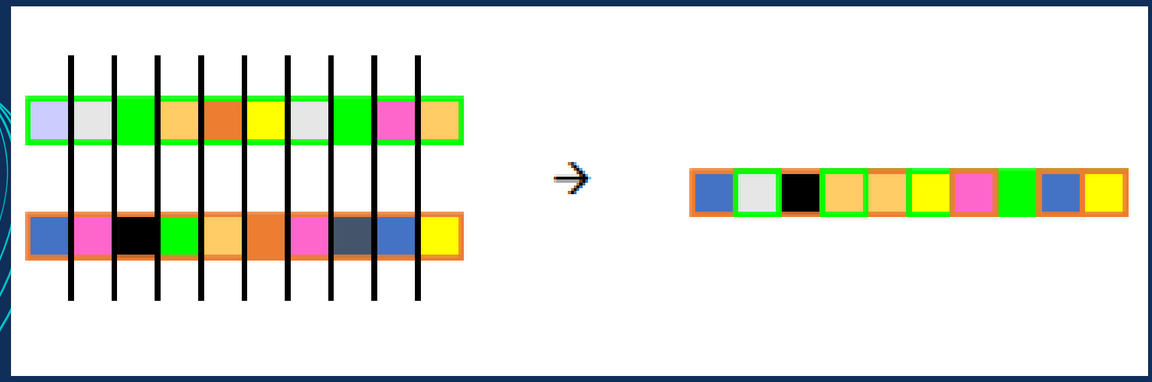
one_cut_cross_over

First 50% from parent 1 and second 50% from parent 2



uniform_cross_over

Alternatively take a gene from relative1 and a gene from relative2



PARENT SELECTION

IF N IS THE SIZE OF THE POPULATION AT EACH ITERATION WE RANDOMLY CHOOSE N / 2 GENOMES AND RETURN THE BEST AMONG THEM BASED ON FITNESS THROUGH THE FUNCTION TOURNAMENT

FITNESS

```python
def compute_fitness(genome):
    list = gen2List(genome)
    repetitions = len(list) - len(set(list))
    return N - len(GOAL - set(list)) , -repetitions
```

WE DECIDED TO MODEL THE FITNESS AS A TUPLE IN WHICH THE FIRST TERM INDICATES HOW FAR THE CURRENT GENOME IS FROM THE SOLUTION WHILE THE SECOND IS A REGULARIZATION TERM THAT FAVORS GENOMES WITH A SMALLER NUMBER OF REPETITIONS

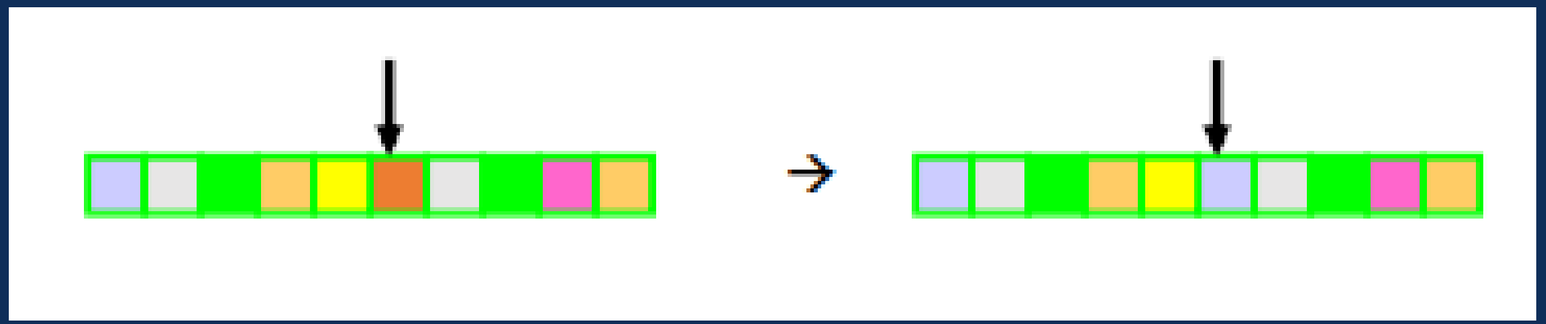
Exploration vs exploitation

A GOOD APPROACH TO FOLLOW WHEN TALKING ABOUT GENETIC ALGORITHMS IS TO DO MORE EXPLORATION AT THE BEGINNING OF THE ALGORITHM IN ORDER TO VISIT AS MANY GENOMES AS POSSIBLE AND THEN AT THE END OF THE ALGORITHM WE REVERSE THE TREND DOING MORE EXPLOITATION FOCUSING ON THE BEST INDIVIDUAL AND CHANGING IT SLIGHTLY

VS

MUTATION

```python
def mutation(g):
    point = random.randint(0, PROBLEM_SIZE - 1)
    return g[:point] + (1 - g[point],) + g[point + 1 :]
```



ARTIFICIAL MUTATION

```
def artificial_mutation(g):
    N=sum(g)
    turn_off = random.randint(1, N)
    count=0
    list_g=list(g)
    for i,el in enumerate(list_g):
        if el:
            count+=1
            if count==turn_off:
                list_g[i]=0
    return mutation(tuple(list_g))
```

At the end of the algorithm we focus only on the best individual and try to optimize it doing exploitation. The artificial_mutation function first sets a gene of the genome to 0 randomly and then calls the mutation function on the new mutated individual

FRANKENSTEIN

Thanks for watching
:-)