GA FOR SET COVERING PROBLEM

- 280117 Giuseppe Piombino https://github.com/Girasolo/computational_intelligence
- 291871 Alessia Leclercq https://github.com/AlessiaLeclercq/ComputationalIntelligence_S291871
- 302294 Leonardo Tredese https://github.com/LeonardoTredese/s302294-computational-intelligence-2022-2023
- 292113 Filippo Cardano https://github.com/Frititati/CI_2022_292113

WORK GROUP

- Idea from the onemax problem:

  each flag correspond to one of the lists generated by the well known function problem

  Genome

```
for genome in [tuple([random.choice([1, 0]) for _ in range(len(ALL_LISTS))]) for _ in range(population_SIZE)]:
    population.append(Individual(genome,fitness_function(genome)))
```

REPRESENTATION

As in the onemax we have used a namedtuple

The fitness function returns a tuple (#coveredEl,-weight), which is assigned to the fitTuple

```
Individual= namedtuple('Individual',['genome','fitTuple'])

def fitness_function(genome: list):
    '''Returns a tuple (number_of_covered_elements, -weigth)'''
    #this fitness function allows the precence of not optimal solution. The hierarchy of them shold be
    # based at first of the #ofCoverdEl and then on the lighter
    num_covered_elements = set()
    weigth = 0
    for (i, list_) in enumerate(ALL_LISTS):
        if genome[i]:
            num_covered_elements.update(list_)
            weigth += len(list_)

    return len(num_covered_elements), -weigth #the reason why weight is negative is we want to minimize it
```

REPRESENTATION

```python
def GA(population, NUM_GENERATIONS, OFFSPRING_SIZE, mutation_rate,POPULATION_SIZE,cleaner_rate):
    start = time.time()                     #to take note of the computation time
    fittestIndividual=copy(population[0])
    for g in range(NUM_GENERATIONS):
        offspring = list()
        for i in range(OFFSPRING_SIZE): #generation of the offSprings
            if random.random() < mutation_rate:
                p = tournament(population)
                if g<NUM_GENERATIONS/cleaner_rate:
                    o= vacuum_cleaner(p.genome)
                else:
                    o = mutation(p.genome)
                #o = mutation(p.genome)

            else:
                p1 = tournament(population)
                p2 = tournament(population)
                o = cross_over(p1.genome, p2.genome)
            offspring.append(Individual(o, fitness_function(o)))
        population += offspring  #offspring added to the population

        #population=deClonizator(population)
        population=list(dict.fromkeys(population))  #deletion of duplicates

        population = sorted(population, key=lambda i: i.fitTuple, reverse=True)[:POPULATION_SIZE]  #cut the population
        if(population[0].fitTuple>fittestIndividual.fitTuple):  #copy the fittest individual
            fittestIndividual=copy(population[0])
```

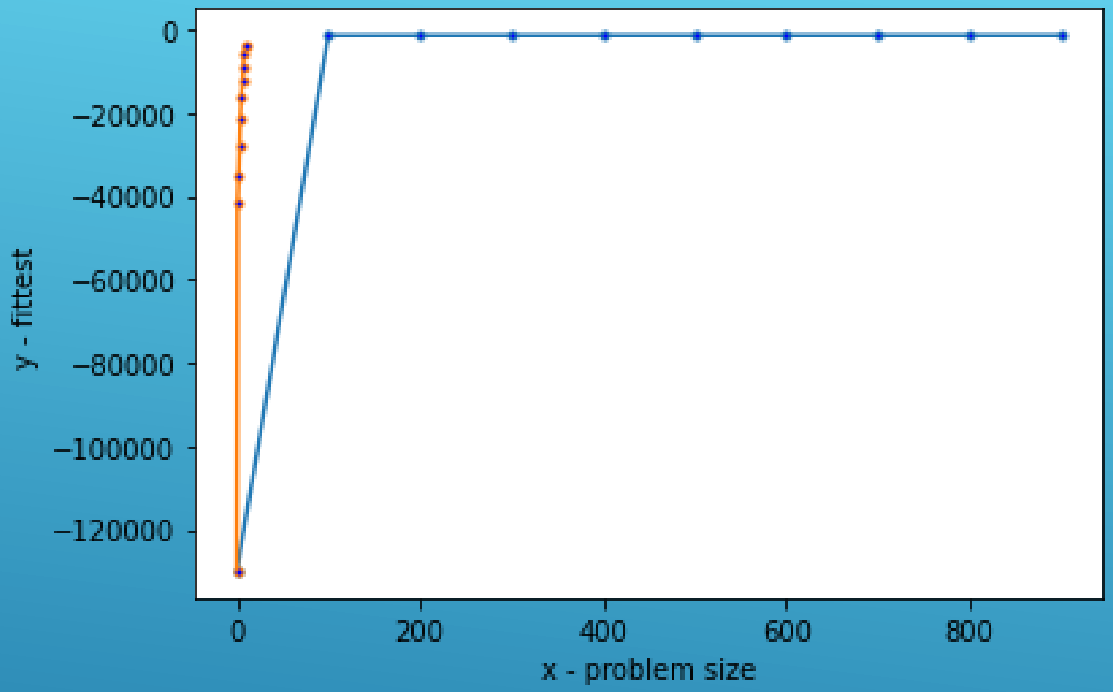
IMPROVEMENTS

```python
def cross_over(g1, g2):
    cut = random.randint(0, len(ALL_LISTS)-1)
    choice=random.randint(0,2)
    if choice==0:
        return g1[:cut] + g2[cut:]
    else:
        return g2[:cut] + g1[cut:]
```

CROSS_OVER

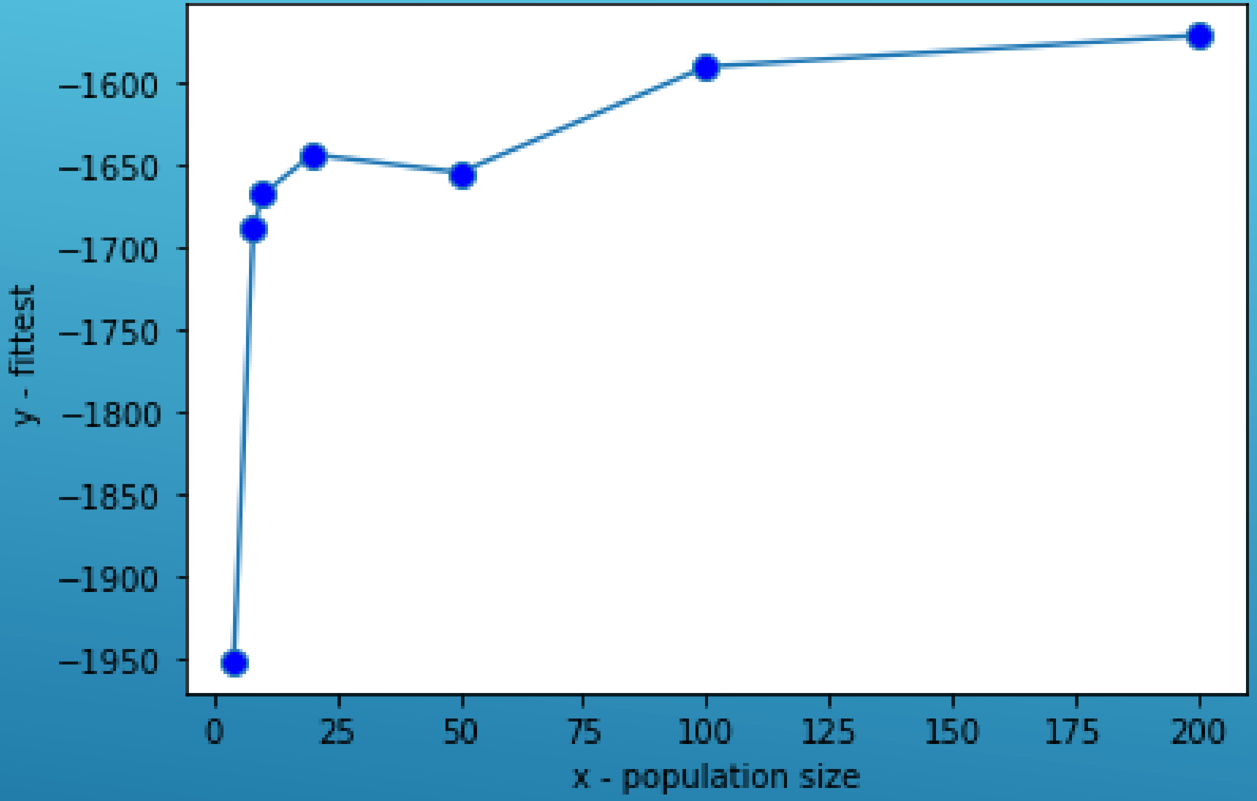
```python
def vacuum_cleaner(g):
    '''takes some weight off by switching off a list'''
    temp_g=g
    for _ in range (random.randint(0,len(ALL_LISTS))):
        point=random.randint(0, len(ALL_LISTS) - 1)
        if temp_g[point] == 1:
            temp_g= temp_g[:point] + (0,) + temp_g[point + 1 :]
    return temp_g
```

VACUUM CLEANER




```
[-129941, -1480, -1460, -1460, -1460, -1460, -1460, -1460, -1460, -1460]
[-129941, -41641, -34870, -27636, -21250, -16377, -12214, -8871, -5626, -3747]
```

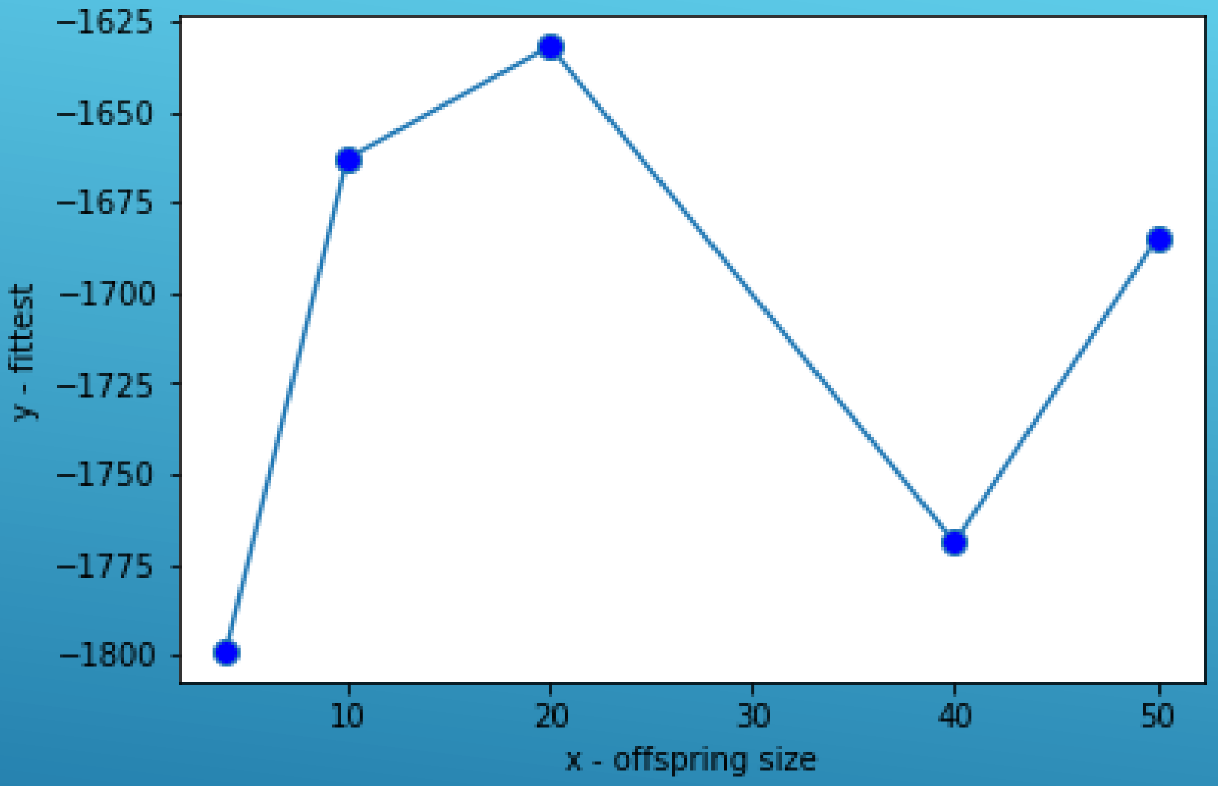
VACUUM CLEANER




```
population_SIZE=[4,8,10,20,50,100,200]
PROBLEM_SIZE=500
OFFSPRING_SIZE=20
mutation_rate=0.3
cleaner_rate=20
NUM_GENERATIONS=1000
```

```
time: 4.03 fittest: (500, -1951)
time: 4.02 fittest: (500, -1688)
time: 3.94 fittest: (500, -1668)
time: 4.08 fittest: (500, -1644)
time: 4.55 fittest: (500, -1655)
time: 5.26 fittest: (500, -1591)
time: 7.01 fittest: (500, -1572)
```

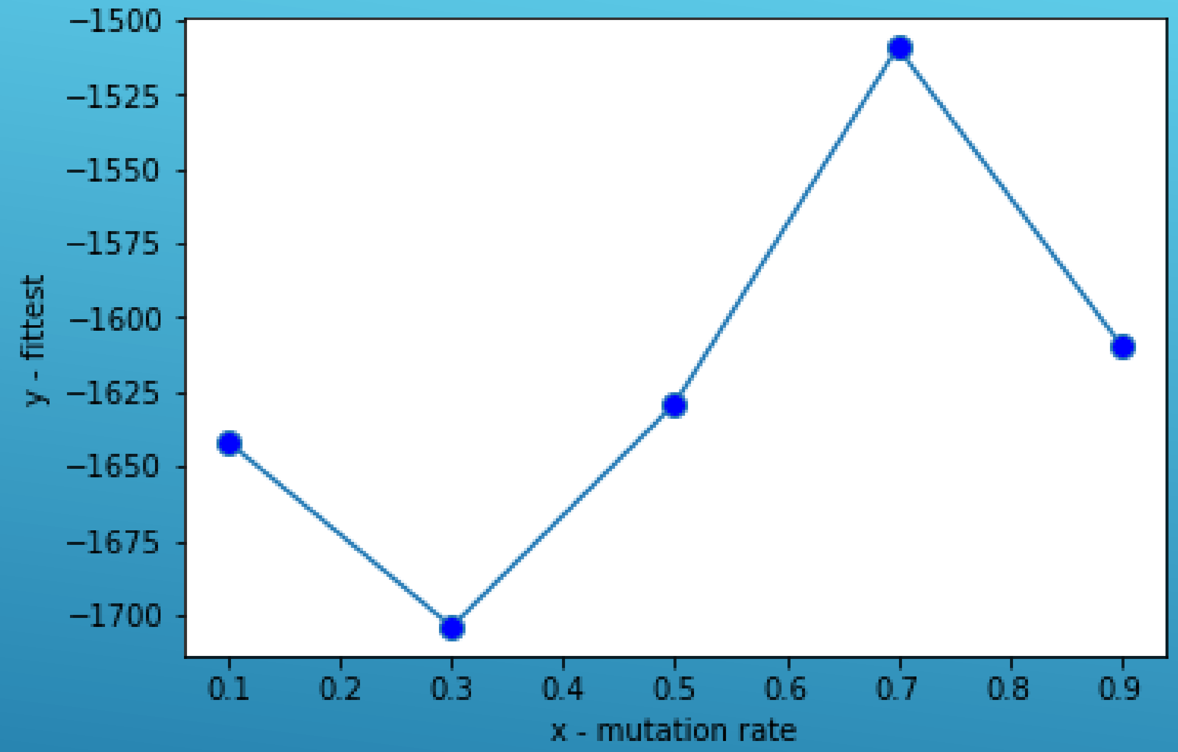
PARAMETER STUDY:POPULATION SIZE




```
population_SIZE=20
PROBLEM_SIZE=500
OFFSPRING_SIZE=[4,10,20,40,50]
mutation_rate=0.3
cleaner_rate=20
NUM_GENERATIONS=1000
```

```
time: 1.41 fittest: (500, -1799)
time: 2.33 fittest: (500, -1663)
time: 4.54 fittest: (500, -1632)
time: 9.63 fittest: (500, -1769)
time: 10.69 fittest: (500, -1685)
```

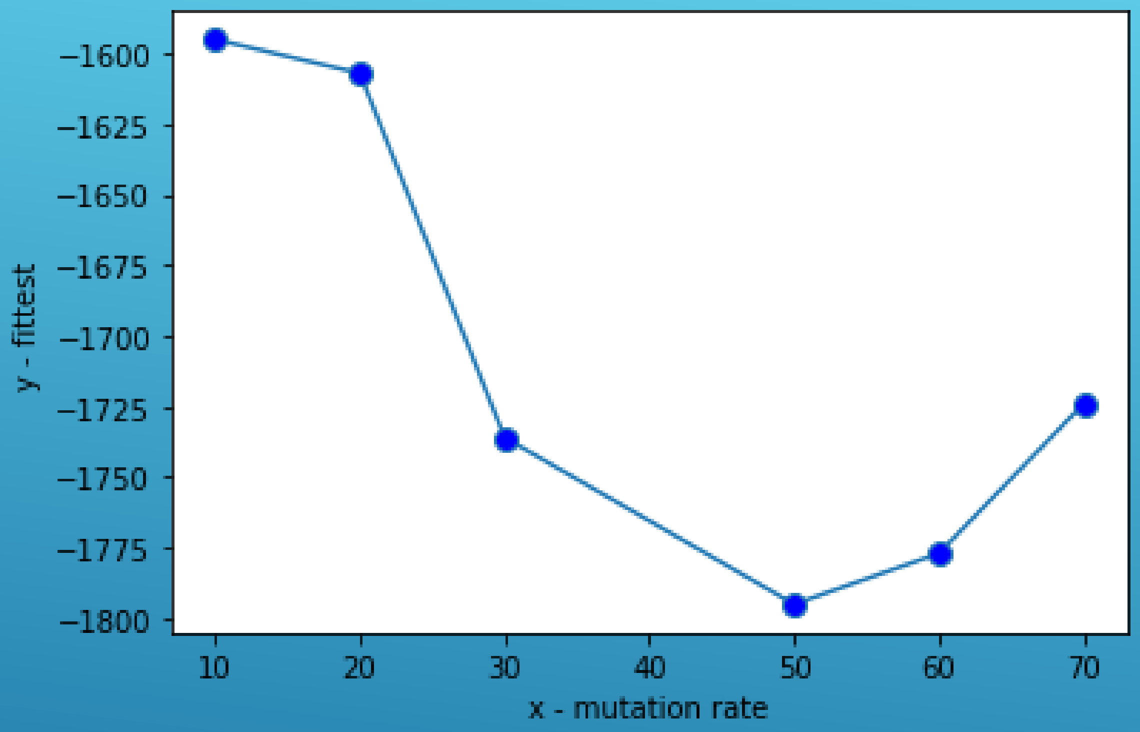
PARAMETER STUDY: OFFSPRING SIZE




```
population_SIZE=20
PROBLEM_SIZE=500
OFFSPRING_SIZE=20
mutation_rate=[0.1,0.3,0.5,0.7,0.9]
cleaner_rate=20
NUM_GENERATIONS=1000
```

```
time: 4.65 fittest: (500, -1642)
time: 4.40 fittest: (500, -1704)
time: 4.67 fittest: (500, -1629)
time: 4.99 fittest: (500, -1509)
time: 5.12 fittest: (500, -1609)
```

PARAMETER STUDY: MUTATION RATE




```
population_SIZE=20
PROBLEM_SIZE=500
OFFSPRING_SIZE=20
mutation_rate=0.7
cleaner_rate=[10,20,30,50,60,70]
NUM_GENERATIONS=1000
```

```
time: 6.10 fittest: (500, -1595)
time: 5.01 fittest: (500, -1607)
time: 4.94 fittest: (500, -1736)
time: 4.39 fittest: (500, -1795)
time: 4.31 fittest: (500, -1777)
time: 4.27 fittest: (500, -1724)
```

PARAMETER STUDY: CLEANER RATE

```
population_SIZE=20
PROBLEM_SIZE=[5,10,20,100,500,1000,2000]
PROBLEM_SIZE_=[5000,10000]
OFFSPRING_SIZE=20
mutation_rate=0.7
cleaner_rate=10
NUM_GENERATIONS=1000
```

```
time: 112.67 fittest: (5000, -26718)
peak memory: 604.85 MiB, increment: 2.73 MiB
time: 271.91 fittest: (10000, -56751)
peak memory: 2073.34 MiB, increment: 3.61 MiB
```

FINAL TRIAL

```
time: 0.26 fittest: (5, -5)
peak memory: 159.83 MiB, increment: -2.07 MiB
time: 0.22 fittest: (10, -11)
peak memory: 159.84 MiB, increment: 0.01 MiB
time: 0.24 fittest: (20, -24)
peak memory: 159.85 MiB, increment: 0.00 MiB
time: 1.14 fittest: (100, -198)
peak memory: 160.15 MiB, increment: 0.25 MiB
time: 5.27 fittest: (500, -1595)
peak memory: 161.92 MiB, increment: 1.06 MiB
time: 12.28 fittest: (1000, -3810)
peak memory: 195.11 MiB, increment: 1.64 MiB
time: 29.81 fittest: (2000, -8789)
peak memory: 309.75 MiB, increment: 4.27 MiB
```

- Use of binary numbers instead of an array of ones and zeros
- …

FURTHER POSSIBLE OPTIMIZATIONS

```python
def GAstrategy2(population, NUM_GENERATIONS, OFFSPRING_SIZE, mutation_rate,POPULATION_SIZE,cleaner_rate):
    start = time.time()                  #to take note of the computation time
    fittestIndividual=copy(population[0])
    for g in range(NUM_GENERATIONS):
        for i in range(OFFSPRING_SIZE):
            o=0
            if random.random() < mutation_rate:
                p = tournament(population)
                if g<NUM_GENERATIONS/cleaner_rate:
                    o= vacuum_cleaner(p.genome)
                else:
                    o = mutation(p.genome)
            if random.random() < 0.5:
                p1 = tournament(population)
                p2 = tournament(population)
                o = cross_over(p1.genome, p2.genome)
            if o:
                population.append(Individual(o, fitness_function(o)))
                population = sorted(population, key=lambda i: i.fitTuple, reverse=True)
                del population[-1]
            #in the strategy2 the population is cut every time we add a new individual

        #population=deClonizator(population)
        population=list(dict.fromkeys(population))  #deletion of duplicates

        if(population[0].fitTuple>fittestIndividual.fitTuple):
            fittestIndividual=copy(population[0])
```

```
#Strategy2
time: 0.25    peak memory: 119.12 MiB fittest: (5, -5)
time: 0.27    peak memory: 119.24 MiB fittest: (10, -14)
time: 0.40    peak memory: 119.26 MiB fittest: (20, -35)
time: 1.93    peak memory: 119.45 MiB fittest: (100, -257)
time: 7.78    peak memory: 126.46 MiB fittest: (500, -1908)
time: 20.20   peak memory: 154.77 MiB fittest: (1000, -4354)
time: 45.22   peak memory: 272.02 MiB fittest: (2000, -9560)
time: 112.67  peak memory: 604.85 MiB fittest: (5000, -26718)
time: 271.91  peak memory: 2073.34 MiB fittest: (10000, -56751)
```

```python
population_SIZE=20
PROBLEM_SIZE_=[5,10,20,100,500,1000,2000]
PROBLEM_SIZE=[5000,10000]
OFFSPRING_SIZE=20
mutation_rate=0.3
cleaner_rate=10
NUM_GENERATIONS=1000
```

ALTERNATIVE STRATEGY: STRATEGY 2