Lab 3 continuation :

Thank you for giving us an opportunity to resubmit the missing section

Submitted by : Abed abo hussien 208517631 Samer Khara'oba 209050202

The ackly function that we emplemented :

- We created 2 ackly function that are functionally the same but one for all algorithms and one for ACO algorithm:

```
def ackly(self, object, target, return_output=True, funct=False):
    a = 20
    b = 0.2
    c = 2*math.pi
    d = 10  # dimentions
    sum_on_x = sum([x ** 2 for x in object.object])
    cos_sum = sum([math.cos(c * x) for x in object.object])
    object.fitness = -a * math.exp(-b * (math.sqrt((1 / d) * sum_on_x))) - math.exp((1 / d) * cos_sum) + a + math.exp(1)
    return object.fitness
def acoackly(self, object, target, return_output=True, funct=False):

    a = 20
    b = 0.2
    c = 2 * math.pi
    d = 10  # dimentions

    cities=target[0]
    sum_on_x = sum([x.x ** 2 for x in cities])
    cos_sum = sum([math.cos(c * x.x) for x in cities])
    object.fitness = -a * math.exp(-b * (math.sqrt((1 / d) * sum_on_x))) - math.exp(
        (1 / d) * cos_sum) + a + math.exp(1)
    return object.fitness
```

- For GA ,simulated annealing ,tabu search and ACO we created a problem set that we used a lot as it's implemented down below :

```
class acly(clark_wright):
    def create_object(self, target_size, target):
        for i in range(10):
            self.object.append(self.character_creation(target_size))

    def character_creation(self, target_size):
        return random.randrange(-32768, 32768) / 1000
```

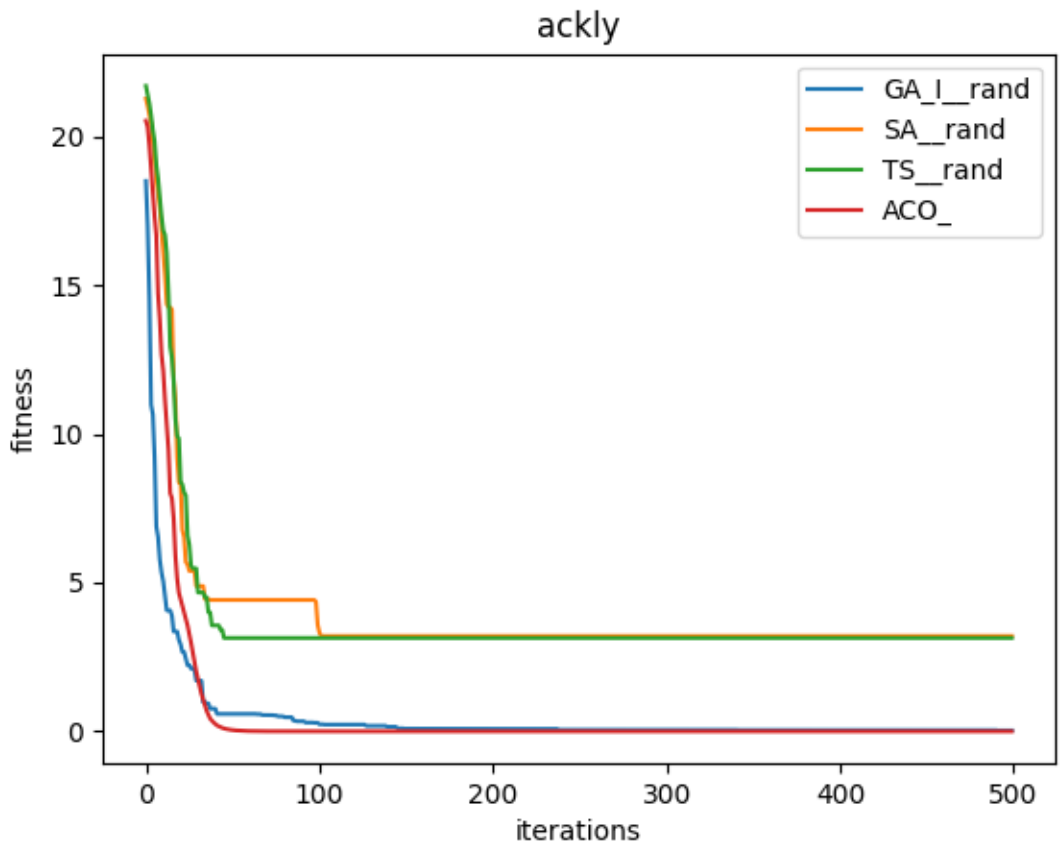
1. Each gene or citizen or solution is defined by a random selection of floats from the range $\in [-32.768, 32.768]$
2. There are 10 of each in every solution, as the dimention was given as equal to 10

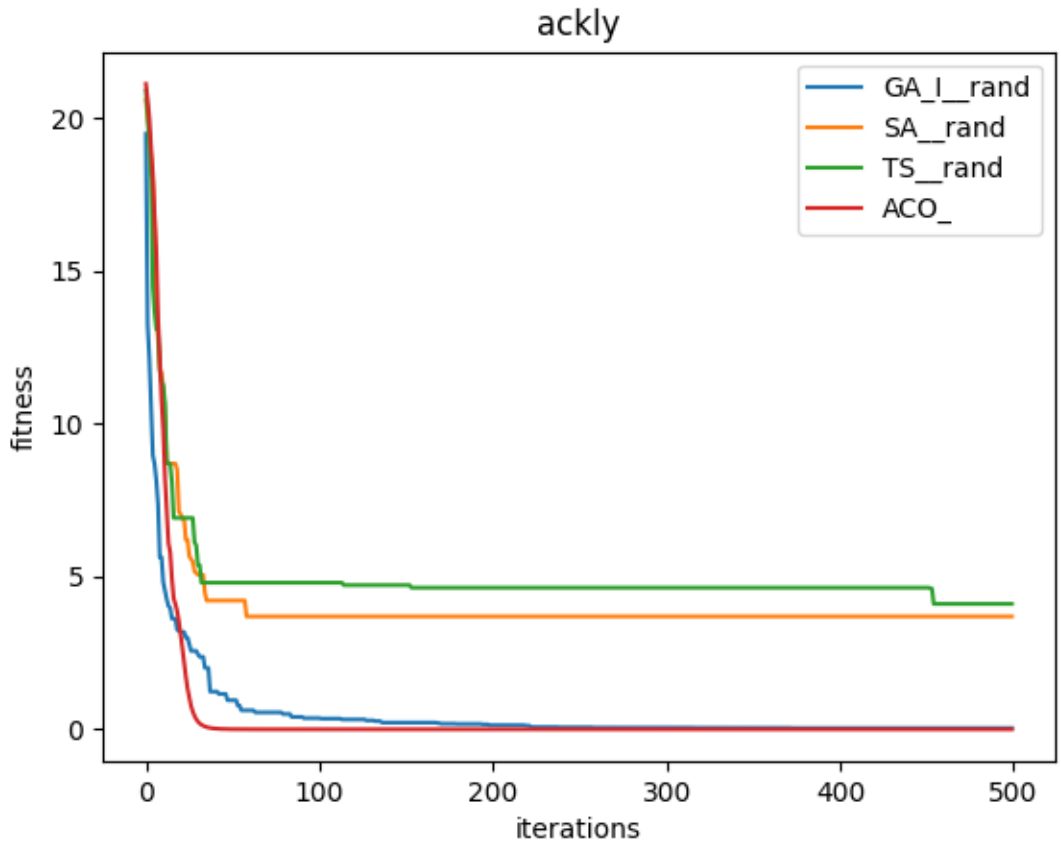
- ACO:
  1. As in ACO we used the same city implementation of ants and put the values solely on its each city's x value , so that every thing works perfectly
  2. Each city has a value $\in [-32,32]$
  3. Aco works on all cities and returns the best solution(path) given the number of cities ,with the fitness value given in Ackley function
  4. The change in solutions is done via arbitrarly changing each element :

```
if self.selection:
    for i in range(10):  # ants
        self.cities[i].x-
=abs(self.cities[i].x/random.randint(1,10)) if
self.cities[i].x>0 else -
abs(self.cities[i].x/random.randint(1,10))
```

  so that we can make the battleground between the algorithms on equal footing .

- Results :
  the results aren't that shocking , as we can see below :





- Output:

```
GA_I__rand
 Best:0.015,0.006,0.001,0.006,0.006,-0.011,0.004,-0.006,0.0,0.017, ,fittness: 0.03992071664783525  Time :  17.5156983  ticks: 17.515002250671387
SA__rand
 Best:0.034,-0.852,-0.002,-0.19,1.912,0.147,0.418,-0.084,0.169,0.925, ,fittness: 3.6962184357023102  Time :  0.29372629999999944  ticks: 0.2932167053222656
TS__rand
 Best:1.104,1.074,-0.28,1.075,0.079,-0.652,1.186,-0.131,0.961,-1.01, ,fittness: 4.106412465945667  Time :  0.3161777999999984  ticks: 0.316270112991333
ACO_
 Best:9,7,4,5,3,8,2,10,6, ,fittness: 4.440892098500626e-16  Time :  0.39334180000000174  ticks: 0.3927946090698242
test results are in the output/ackley folder
press any key then enter to return to main menu otherwize type exit:
```

- Analysis :
  - as we can see ACO and GA don’t get stuck on local maxima .
  - simulated annealing and tabu search do , even if we continue the search they will still get stuck at local minima
- GUI :

```
1: manual
2:automatic test
3:ackly test:3
```

To redo the tests in this portion you can press 3

To redo the tests in the first report press 2

To check manually select 1

- output folder contains all the individual tests from graphs to text that describes each algorithm

Script of this section :

```
fitness_arr,iter_array=[],[]

for select_generator in range(0, 1):
    # cities, cost_matrix, dimentions, capacity = get_sets_from_files(inputs[select_input])
    cities, cost_matrix, dimentions, capacity = ackley()
    GA_TARGET = [cities, cost_matrix, dimentions, capacity]
    target_size = len(cities)

    for mutation in range(1,2):

        print("GA_I_"+heuristics[select_generator]+"_"+mutation_index[mutation])

        sol=GA_LAB1(GA_TARGET, target_size, GA_POPSIZE, problem_set, 2, "ackly", 3,
                                1, mutation, Gene_dist,max_iter=max_iter)
        fitness,iteration=sol.solve()
        fitness_arr.append(fitness)
        iter_array.append(iteration)
        names.append("GA_I_"+heuristics[select_generator]+"_"+mutation_index[mutation])

        print("SA_"+heuristics[select_generator]+"_"+mutation_index[mutation])

        sol = algo[3](GA_TARGET, target_size, GA_POPSIZE, problem_set, "ackly", max_iter, mutation)
        fitness, iteration = sol.solve()
        fitness_arr.append(fitness)
        iter_array.append(iteration)
        names.append("SA_"+heuristics[select_generator]+"_"+mutation_index[mutation])

        print("TS_"+heuristics[select_generator]+"_"+mutation_index[mutation])

        sol = algo[4](GA_TARGET, target_size, GA_POPSIZE, problem_set, "ackly", max_iter, mutation)
        fitness, iteration = sol.solve()
        fitness_arr.append(fitness)
        iter_array.append(iteration)
        names.append("TS_"+heuristics[select_generator]+"_"+mutation_index[mutation])

    print("ACO_" + heuristics[select_generator] )

    sol = algo[2](GA_TARGET, target_size, GA_POPSIZE, problem_set, "acoackly", max_iter,selection=True)
```