AI Lab 4 part 1

CoEvolution

Submitted BY :

Abed Abo Hussien 208517631

Samer Kharaoba 209050202

First of all how we implemented the network :

1. Create object: creates a network (couples of places [3,5] for example) with up to 10% more comparators than the optimal configuration .
2. A network of 4 element would be : ([1,3],[2,4],[3,2],[1,4]) for example
3. Solve_network: given a set of input check the output of the network
4. Check solution: checks if the given set is ordered correctly
5. Compare : takes 2 places in the network ,compares them and swaps them if there is a need to do that
6. Get depth, depth check, get depth at index: all work together to find the depth of the current network
7. Apply : applies a set of inputs on the network and by that calculates it's fitness
8. Valid insertion : some insertions cannot be done such as [1,3],[1,3] in a row defeats the purpose of doing the first [1,3] comparison.

```
class network(DNA):
    def __init__(self):
        super(network, self).__init__()
        depth = 0

    def create_object(self, target_size, target, options=None):...

    def character_create(self, target_size, options):...

    def solve_network(self, set, target_size):...

    def check_solution(self, set, target_size):...

    def apply(self, sets, target_size, target):...

    def compare(self, first, second, set):...

    def get_depth(self):...

    def get_depth_at_index(self, i):...

    def __str__(self):...

    def depth_check(self, lista, items):...

    def valid_insertion(self, ipos, options):...
```

Now that we explained what the network looks like we will now first explain section 1 in the report :

Implementation of the fitness function :

1. Go over all given sets to sort
2. Give each one a number of networks that was tested on it
3. Give each one the number of networks that successfully reordered the set
4. After all the sets have been checked get the percentage of failure denoted by :
   - (1-sum_of_sets/len(sets))
5. Check the depth of that network and make it affect 50% of the fitness

```
def apply(self, sets, target_size, target):
    all_sets_sorted = []
    for set in sets:
        set.diversity = 0
        set.networks_tested = 0
        all_sets_sorted.append(self.solve_network(set, target_size))
    sum_of_sets = 0
    for set in sets:
        sum_of_sets += set.diversity
    # we want the lower number
    depth = self.get_depth()
    factor = len(depth) / target[2]
    if factor != 1.0:
        factor = 100 * factor
    else:
        factor = 0
    self.fitness = factor + 100 * (1 - sum_of_sets / len(sets))
    return all_sets_sorted
```

Implementation of fitness on the parasites :

The exact opposite of the networks fitness :

```
    set.fitness=100*set.diversity/set.networks_tested if set.networks_tested else set.fitness
```

Compare:

```
def compare(self, first, second, set):
    if set[first] <= set[second]:
        return
    else:
        set[first], set[second] = set[second], set[first]
```

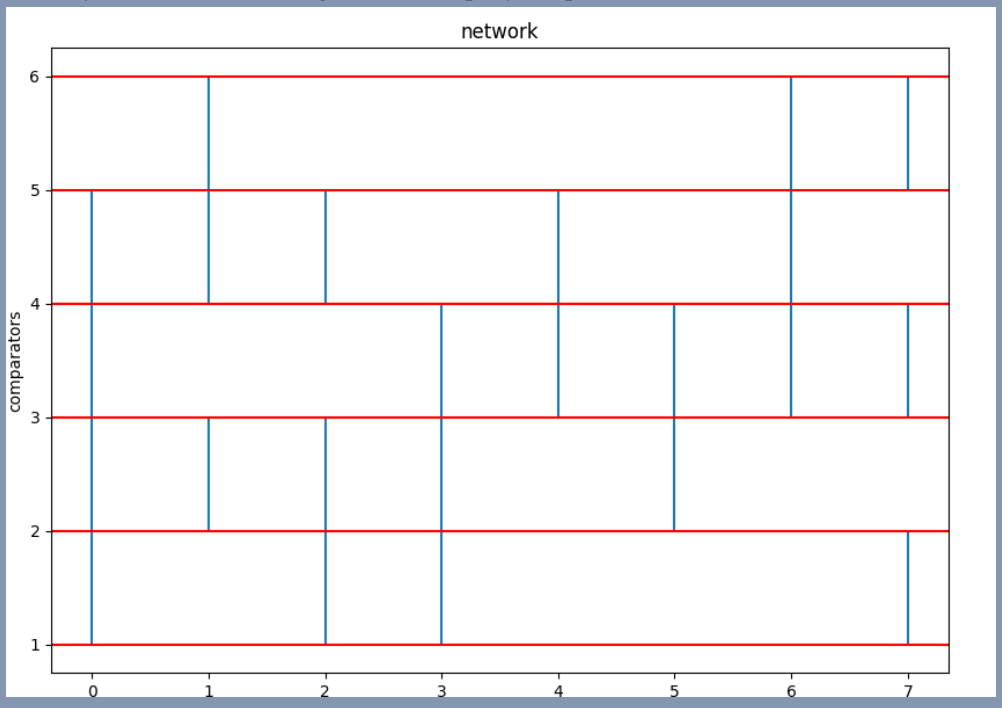
Both of those fitness functions are implemented in our fitness class , so that when we mutate one of them we can use those functions here:

```
class fitness_selector:
    def __init__(self):
        self.select = {0: self.sets_fitness,1:self.networks_fitness}

    def sets_fitness(self, object, target, target_size,networks,sets):

        for network in networks:
            network.solve_network(object,target_size)

        object.fitness=100*object.diversity/object.networks_tested

    def networks_fitness(self, object, target, target_size,networks,sets):
        object.apply(_sets,target_size,target)
        return object.fitness
```

The algorithm returns a depth based output :

```
Best:depth 1: (1,4)(3,6)
depth 2: (2,3)(1,6)
depth 3: (1,4)(2,6)(3,5)
depth 4: (2,4)(3,6)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 79.5  Time :  1.3747559000000003  ticks: 1.3753232955932617
```

And by the end of the algorithm displays a plot of the network such as :



Output for input of size 6 :

```
 ,fittness: 17.000000000000004  Time :  9.4877256  ticks: 9.487738847732544
Best:depth 1: (4,5)(1,6)
depth 2: (5,6)(2,3)(1,4)
depth 3: (2,4)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 14.000000000000002  Time :  10.3877519  ticks: 10.3879075050354
Best:depth 1: (2,4)(1,3)(5,6)
depth 2: (2,5)(1,4)
depth 3: (2,5)(4,6)
depth 4: (3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 42.50000000000001  Time :  11.2853888  ticks: 11.285508871078491
Best:depth 1: (2,6)(4,5)(1,3)
depth 2: (1,4)
depth 3: (2,4)(3,6)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 5.500000000000005  Time :  12.275672100000001  ticks: 12.275930404663086
Best:depth 1: (4,5)(1,6)
depth 2: (5,6)(2,3)
depth 3: (1,3)(2,4)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 12.5  Time :  13.096205999999999  ticks: 13.096761465072632
Best:depth 1: (2,6)(4,5)(1,3)
depth 2: (1,4)
depth 3: (2,4)(3,6)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 19.499999999999996  Time :  14.0430141  ticks: 14.043230533599854
Best:depth 1: (2,6)(4,5)(1,3)
depth 2: (1,4)(2,5)
depth 3: (2,4)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 6.000000000000005  Time :  14.9889693  ticks: 14.98881220817566
Best:depth 1: (1,6)(2,5)
depth 2: (1,3)(4,5)
depth 3: (1,4)(3,6)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 9.499999999999996  Time :  15.9031908  ticks: 15.90315842628479
Best:depth 1: (2,5)(1,6)
depth 2: (5,6)(2,3)(1,4)
depth 3: (4,5)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 13.0  Time :  16.7537313  ticks: 16.753381490707397
Best:depth 1: (4,5)(2,3)
depth 2: (1,4)(3,6)
depth 3: (4,5)(2,3)
depth 4: (4,6)(3,5)(1,2)
depth 5: (1,2)(3,4)(5,6)
 ,fittness: 14.000000000000002  Time :  17.5768325  ticks: 17.577037572860718
```

-Changes done to the genetic algorithm :

1. It now holds 2 populations , sorted networks and parasites in our code we still call the parasites self.population
2. Here we initiate both populations :

```
def init_population(self):
    super(C_genetic_algorithem, self).init_population()
    self.init_networks()
    self.populations=[self.population,self.sorting_networks]
def init_networks(self):
    # todo:T initiate networks here
    #  1.T update self.sorting_networks
    #  2.T defin new target and targetsize
    for i in range(self.num_networks):
        temp=self.problem_spec2()
        temp.create_object( self.target_size, self.target,self.options)
        self.sorting_networks.append(temp)
```

3. We used 500 sorted networks , generated randomly

- Changes done to the mutation operators :

1. Here we don't select arbitrarily we get to know which are in the current,previous and next depth so that we don't chose them :

```
def random_mutate(self, target_size, member, options):
    ipos = random.randint(0, target_size - 1)
    new_options = member.valid_insertion(ipos, options)
    delta = member.character_create(len(new_options),new_options)
    member.object = member.object[:ipos] + [delta] + member.object[ipos + 1:]
```

2. Modify operator : basically choses a comparator and stretches it or the opposite like :

```
def modify(self, target_size, member, options):
    # modify
    size = len(member.object)
    ipos = random.randint(0, size - 1)
    bigger = random.choice([0, 1])
    up_down = random.choice([0, 1])
    new = member.object[ipos]
    if bigger:
        if up_down:
            new[0] -= 1 if new[0] > 1 else 0
        else:
            new[1] += 1 if new[1] < target_size else 0
    else:
        if up_down:
            new[0] += 1 if new[1] > new[0] + 1 else 0
        else:
            new[1] -= 1 if new[0] + 1 < new[1] else 0
```

-Cross operation wasn't changed , but we chose the single cross.

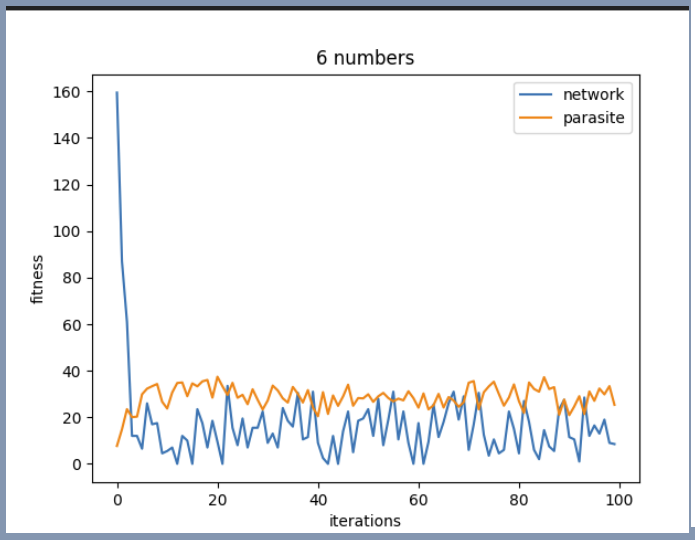
Section 2 :

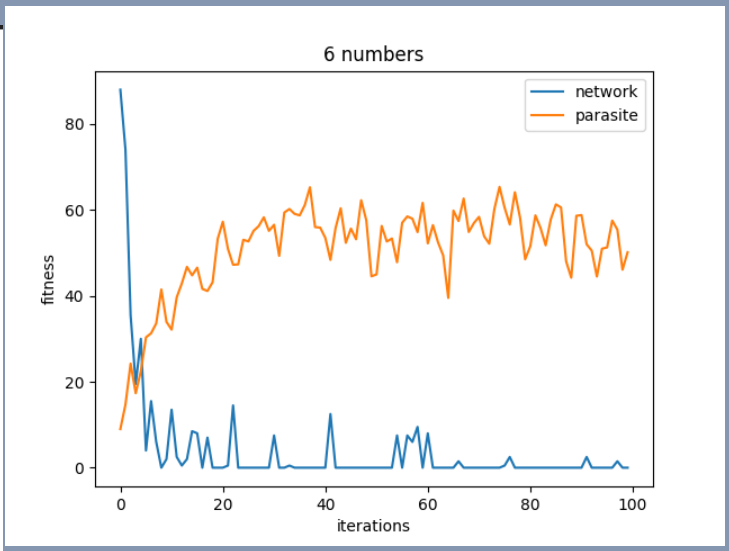
Results :

On the 6 numbers we got the following results :

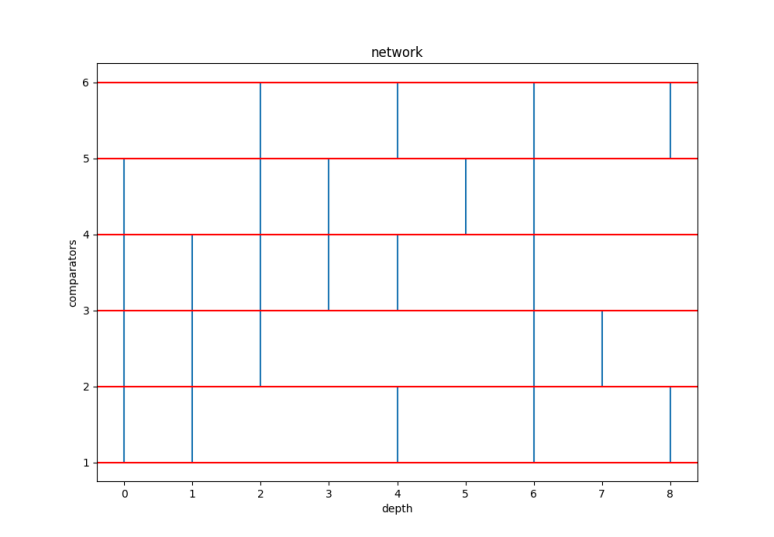
Note the the bar below isn't depth , this was a typo :

Given one graph over 100 iterations :





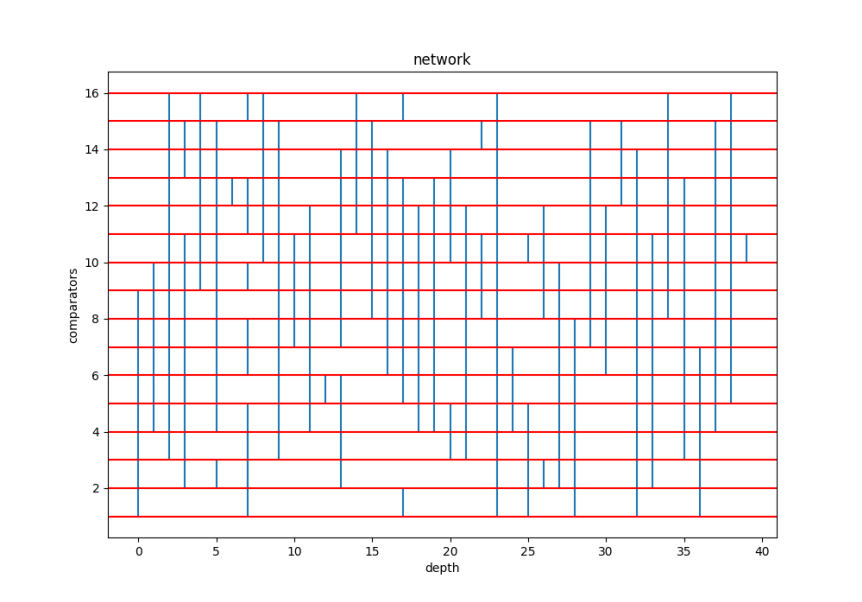
Result:



On the 16 numbers we got the following results :

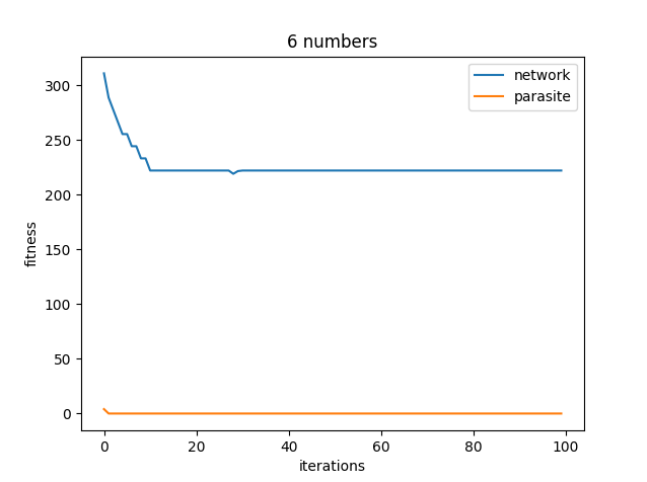
Note the the bar below isn't depth , this was a typo :

Given one graph over 100 iterations :



these are for the 16 numbers :

we can clearly see that our algorithm didn't succeed to get the network to a fitness of 0 because the solution that we got in the results is composed of 61 comparitors and isn't the optimal one , although the parasites seem dormant , they aren't but the algorithm falls to a local minima pretty quickly :



Section 3 :

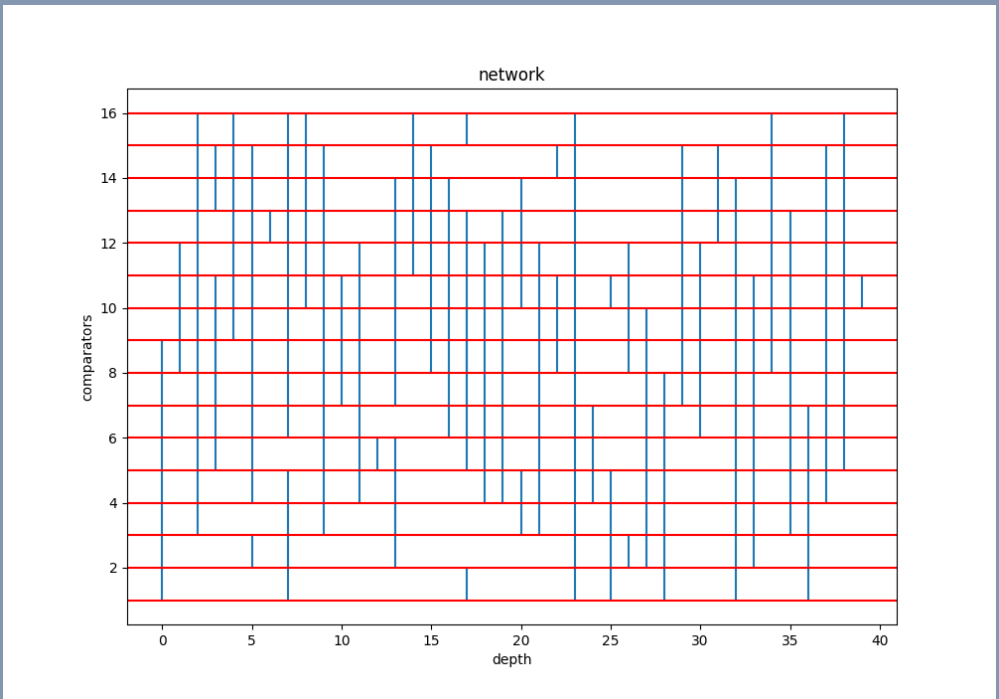
In the following results we can see Strategy Recycling effect as the top 10 solutions are mostly recycled solutions mashed together at the top as they are the top performers :

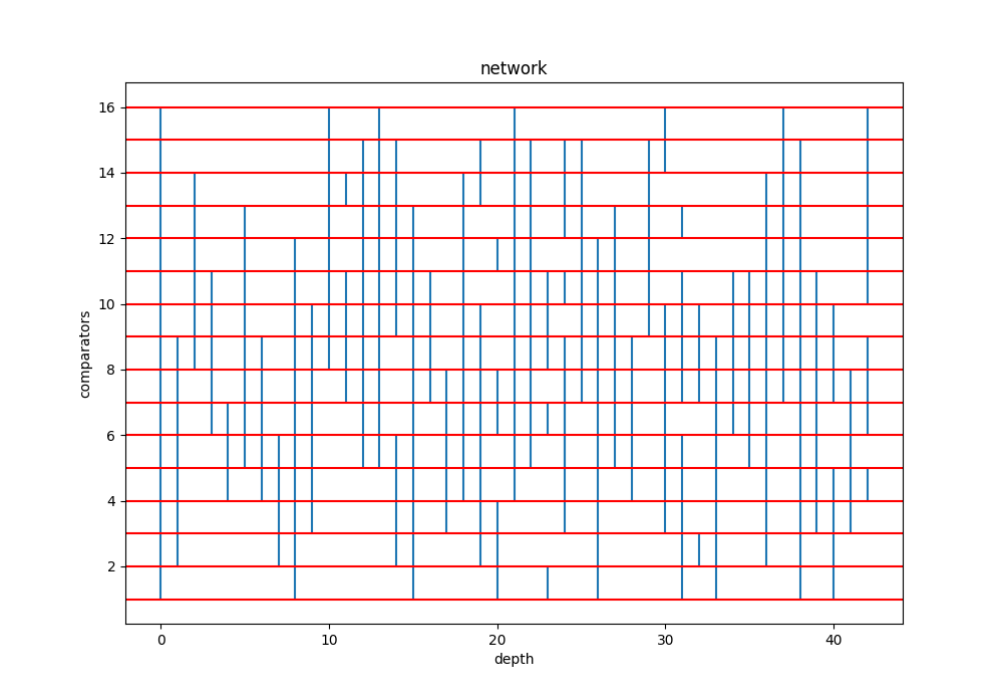
```
depth 1: (2,6)(1,3)
 depth 2: (4,6)(1,2)(3,5)
 depth 3: (3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (2,6)(1,3)
 depth 2: (4,6)(1,2)(3,5)
 depth 3: (3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (1,3)(2,4)
 depth 2: (4,6)(1,2)(3,5)
 depth 3: (3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (2,6)(1,3)
 depth 2: (4,6)(1,2)(3,5)
 depth 3: (3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (2,6)(1,3)
 depth 2: (4,6)(1,2)(3,5)
 depth 3: (3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (2,6)(1,4)
 depth 2: (4,6)(1,2)
 depth 3: (1,5)(3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (2,6)(1,3)
 depth 2: (4,6)(1,2)(3,5)
 depth 3: (3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (2,6)(1,4)
 depth 2: (4,6)(1,2)
 depth 3: (1,5)(3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (2,6)(1,4)
 depth 2: (4,6)(1,2)
 depth 3: (1,5)(3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)

depth 1: (1,3)(2,4)
 depth 2: (4,6)(1,2)(3,5)
 depth 3: (3,6)(2,4)
 depth 4: (4,5)(2,3)
 depth 5: (3,4)(5,6)(1,2)
```

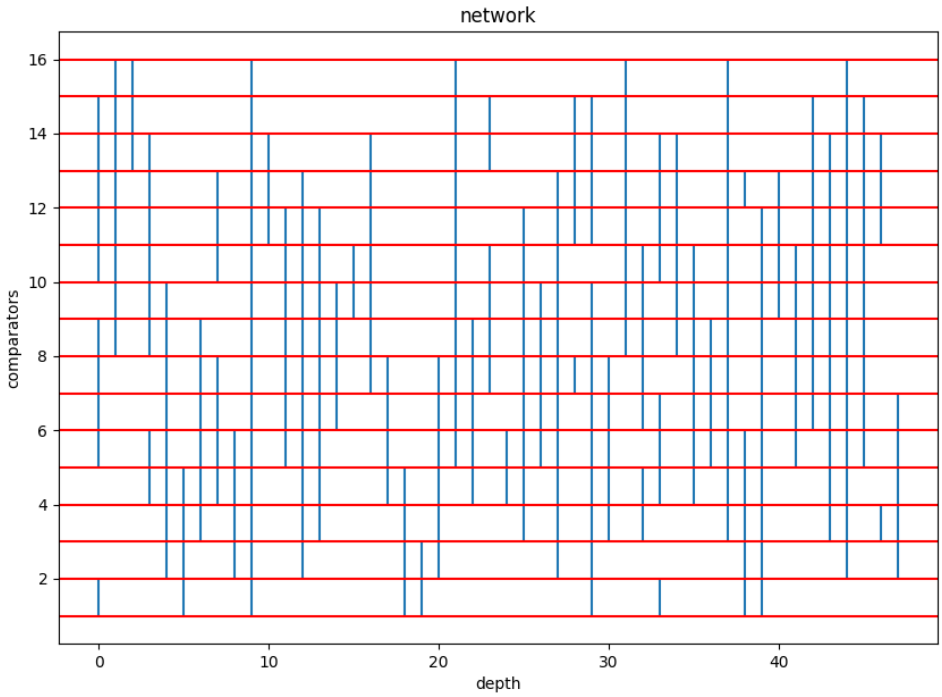
```
depth 1: (1,9)(4,10)(3,16)(2,6)(13,15)(5,11)
 depth 2: (9,16)(10,11)(2,3)(4,15)(12,13)
 depth 3: (9,10)(1,4)(15,16)(2,5)(11,13)(6,8)
 depth 4: (10,16)(3,15)(7,11)(4,12)(5,6)
 depth 5: (3,4)(10,13)(2,6)(7,14)(11,16)(8,15)
 depth 6: (6,14)(8,10)(1,2)(5,13)(15,16)(7,9)(4,12)
 depth 7: (4,13)(11,12)(3,5)(10,14)
 depth 8: (3,12)(8,11)(14,15)(1,16)(4,7)
 depth 9: (1,5)(10,11)(8,12)(2,3)
 depth 10: (2,10)(1,8)(7,15)(6,12)
 depth 11: (12,15)(1,14)(2,11)(8,16)(3,13)
 depth 12: (1,7)(4,15)(5,16)(10,11)

depth 1: (1,9)(4,10)(3,16)(2,6)(13,15)(5,11)
 depth 2: (9,16)(10,11)(2,3)(4,15)(12,13)(7,14)
 depth 3: (1,4)(15,16)(2,5)(11,13)(6,8)
 depth 4: (10,16)(3,15)(7,11)(4,12)(5,6)
 depth 5: (3,4)(10,13)(2,6)(7,14)(11,16)(8,15)
 depth 6: (6,14)(8,10)(1,2)(5,13)(15,16)(7,9)(4,12)
 depth 7: (4,13)(11,12)(3,5)(10,14)
 depth 8: (3,12)(8,11)(14,15)(1,16)(4,7)
 depth 9: (1,5)(10,11)(8,12)(2,3)
 depth 10: (2,10)(1,8)(7,15)(6,12)
 depth 11: (12,15)(1,14)(2,11)(8,16)(3,13)
 depth 12: (1,7)(4,15)(5,16)(10,11)

depth 1: (1,9)(4,10)(3,16)(2,6)(13,15)(5,11)
 depth 2: (9,16)(10,11)(2,3)(4,8)(12,13)(7,14)
 depth 3: (1,4)(15,16)(2,5)(11,13)(6,8)
 depth 4: (10,16)(3,15)(7,11)(4,12)(5,6)
 depth 5: (3,4)(10,13)(2,6)(7,14)(11,16)(8,15)
 depth 6: (6,14)(8,10)(1,2)(5,13)(15,16)(7,9)(4,12)
 depth 7: (4,13)(11,12)(3,5)(10,14)
 depth 8: (3,12)(8,11)(14,15)(1,16)(4,7)
 depth 9: (1,5)(10,11)(8,12)(2,3)
 depth 10: (2,10)(1,8)(7,15)(6,12)
 depth 11: (12,15)(1,14)(2,11)(8,16)(3,13)
 depth 12: (1,7)(4,15)(5,16)(10,11)
```

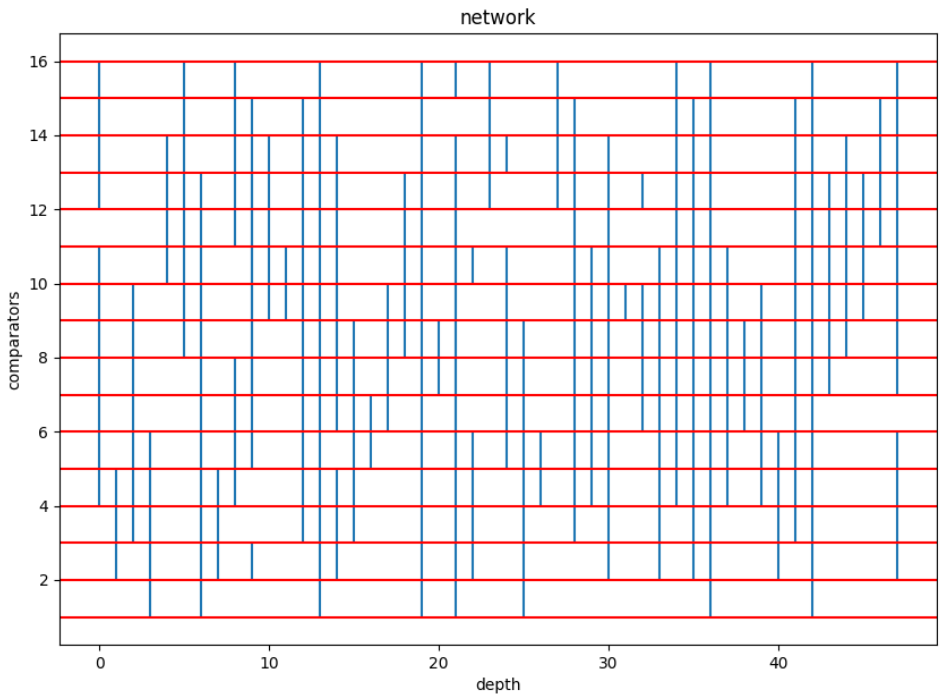
```
depth 1: (1,2)(10,14)(5,9)(12,15)(8,16)
 depth 2: (13,16)(11,15)(4,6)(2,10)(1,5)(3,9)
 depth 3: (10,13)(4,8)(2,6)(1,16)(11,14)(5,12)
 depth 4: (2,13)(3,12)(6,10)(9,11)(7,14)(4,8)(1,5)
 depth 5: (1,3)(2,8)(5,16)(4,9)(7,11)(13,15)
 depth 6: (4,6)(3,12)(5,10)(2,13)(11,15)(7,8)
 depth 7: (11,15)(1,10)(3,8)
 depth 8: (8,16)(6,11)(3,5)(4,7)(1,2)(10,14)
 depth 9: (8,14)(4,11)(5,9)(3,16)(12,13)(1,6)
 depth 10: (1,12)(9,13)(5,11)(6,15)(3,14)(2,16)
 depth 11: (5,15)(11,14)(3,4)(2,7)

depth 1: (1,2)(10,14)(5,9)(12,15)(8,16)
 depth 2: (13,16)(11,15)(4,6)(2,10)(7,9)
 depth 3: (3,9)(10,13)(4,8)(2,6)(1,16)(11,14)(5,12)
 depth 4: (2,13)(3,12)(6,10)(9,11)(7,14)(4,8)(1,5)
 depth 5: (1,3)(2,8)(5,16)(4,9)(7,11)(13,15)
 depth 6: (4,6)(3,12)(5,10)(2,13)(11,15)(7,8)
 depth 7: (11,15)(1,10)(3,8)
 depth 8: (8,16)(6,11)(3,5)(4,7)(1,2)(10,14)
 depth 9: (8,14)(4,11)(5,9)(3,16)(12,13)(1,6)
 depth 10: (1,12)(9,13)(5,11)(6,15)(3,14)(2,16)
 depth 11: (5,15)(11,14)(3,4)(2,7)

depth 1: (1,2)(10,14)(5,9)(12,15)(8,16)
 depth 2: (13,16)(11,15)(4,6)(2,10)(1,5)(3,9)
 depth 3: (10,13)(4,8)(2,6)(1,16)(11,14)(5,12)
 depth 4: (2,13)(3,12)(6,10)(9,11)(7,14)(4,8)(1,5)
 depth 5: (1,3)(2,8)(5,16)(4,9)(7,11)(13,15)
 depth 6: (4,6)(3,12)(5,10)(2,13)(11,15)(7,8)
 depth 7: (11,15)(1,10)(3,8)
 depth 8: (8,16)(6,11)(3,5)(4,7)(1,2)(10,14)
 depth 9: (8,14)(4,11)(5,9)(3,16)(12,13)(1,6)
 depth 10: (1,12)(9,13)(5,11)(6,15)(3,14)(2,16)
 depth 11: (5,15)(11,14)(3,4)(2,7)

depth 1: (1,2)(10,14)(5,9)(12,15)(8,16)
 depth 2: (13,16)(8,14)(4,6)(2,10)(1,5)(3,9)
 depth 3: (10,13)(4,8)(2,6)(1,16)(11,14)(5,12)
 depth 4: (2,13)(3,12)(6,10)(9,11)(7,14)(4,8)(1,5)
 depth 5: (1,3)(2,8)(5,16)(4,9)(7,11)(13,15)
 depth 6: (4,6)(3,12)(5,10)(2,13)(11,15)(7,8)
 depth 7: (11,15)(1,10)(3,8)
 depth 8: (8,16)(6,11)(3,5)(4,7)(1,2)(10,14)
 depth 9: (8,14)(4,11)(5,9)(3,16)(12,13)(1,6)
 depth 10: (1,12)(9,13)(5,11)(6,15)(3,14)(2,16)
 depth 11: (5,15)(11,14)(3,4)(2,7)

depth 1: (3,14)(4,16)(9,15)(2,7)(8,12)
 depth 2: (13,16)(2,10)(12,15)(6,7)(9,11)(3,8)(1,14)
 depth 3: (3,13)(2,6)(8,9)(11,14)(5,12)
 depth 4: (2,13)(3,12)(6,10)(9,11)(7,14)(4,8)(1,5)
 depth 5: (1,3)(2,8)(5,16)(4,9)(7,11)(13,15)
 depth 6: (4,6)(3,12)(5,10)(2,13)(11,15)(7,8)
 depth 7: (11,15)(1,10)(3,8)
 depth 8: (8,16)(6,11)(3,5)(4,7)(1,2)(10,14)
 depth 9: (8,14)(4,11)(5,9)(3,16)(12,13)(1,6)
 depth 10: (1,12)(9,13)(5,11)(6,15)(3,14)(2,16)
 depth 11: (5,15)(11,14)(3,4)(2,7)

depth 1: (8,11)(10,14)(4,5)(15,16)(3,7)(1,2)
 depth 2: (1,10)(8,12)(6,7)(9,11)(15,16)
 depth 3: (10,13)(4,8)(2,6)(1,16)(11,14)(5,12)
 depth 4: (2,13)(3,12)(6,10)(9,11)(7,14)(4,8)(1,5)
 depth 5: (1,3)(2,8)(5,16)(4,9)(7,11)(13,15)
 depth 6: (4,6)(3,12)(5,10)(2,13)(11,15)(7,8)
 depth 7: (11,15)(1,10)(3,8)
 depth 8: (8,16)(6,11)(3,5)(4,7)(1,2)(10,14)
 depth 9: (8,14)(4,11)(5,9)(3,16)(12,13)(1,6)
 depth 10: (1,12)(9,13)(5,11)(6,15)(3,14)(2,16)
 depth 11: (5,15)(11,14)(3,4)(2,7)
```

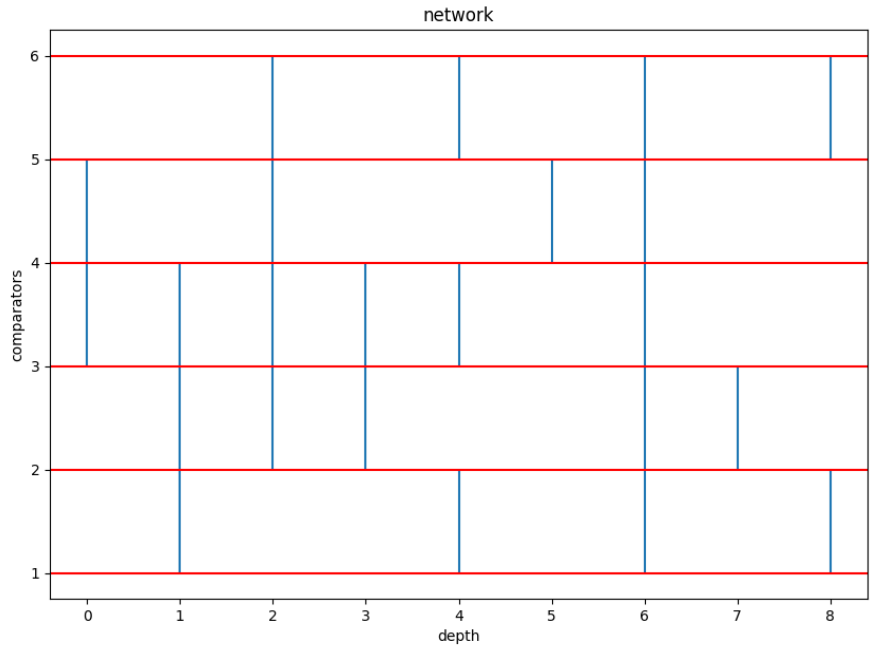
As are the sorting networks :

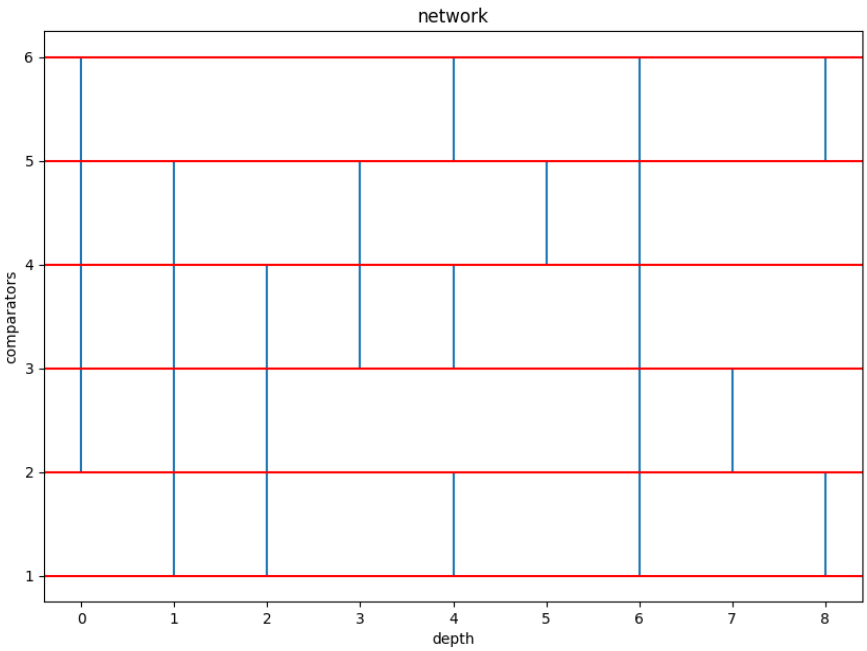


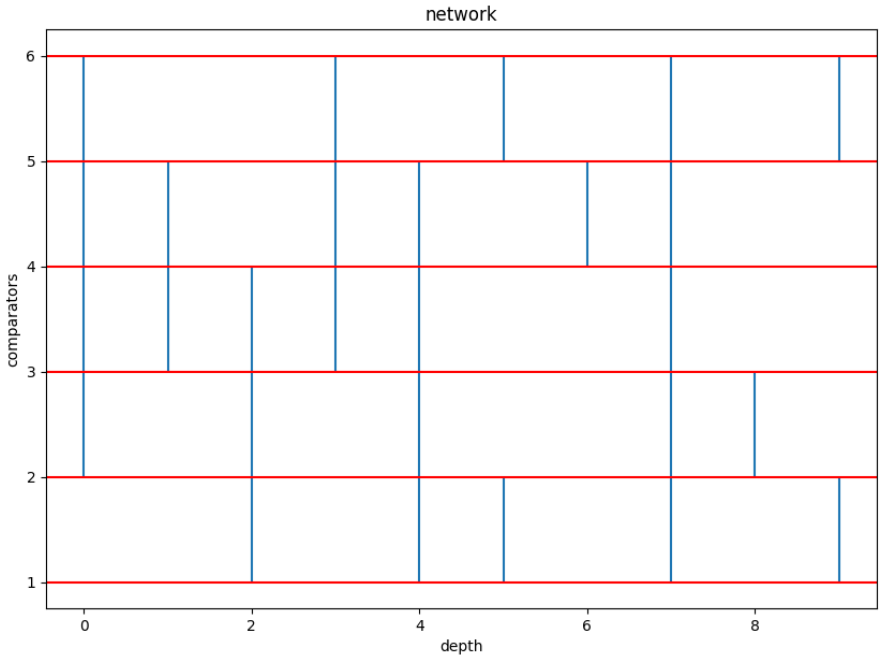


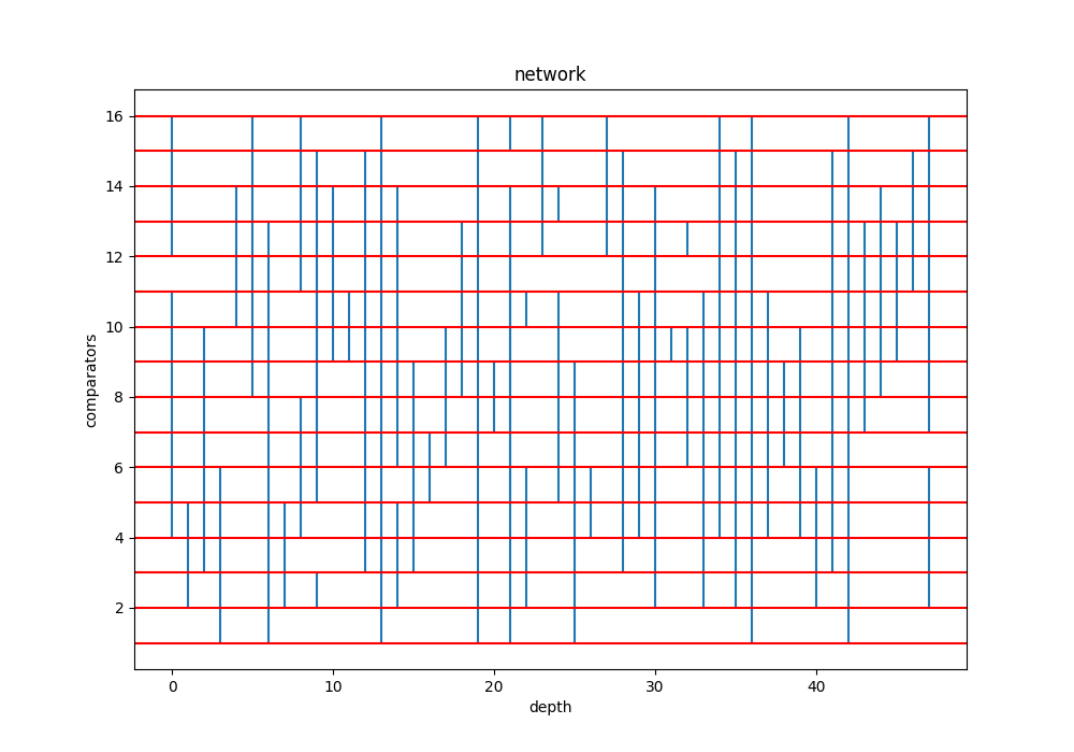

network
comparators
depth
16
14
12
10
8
6
4
2
0
10
20
30
40


network
comparators
depth
16
14
12
10
8
6
4
2
0
10
20
30
40

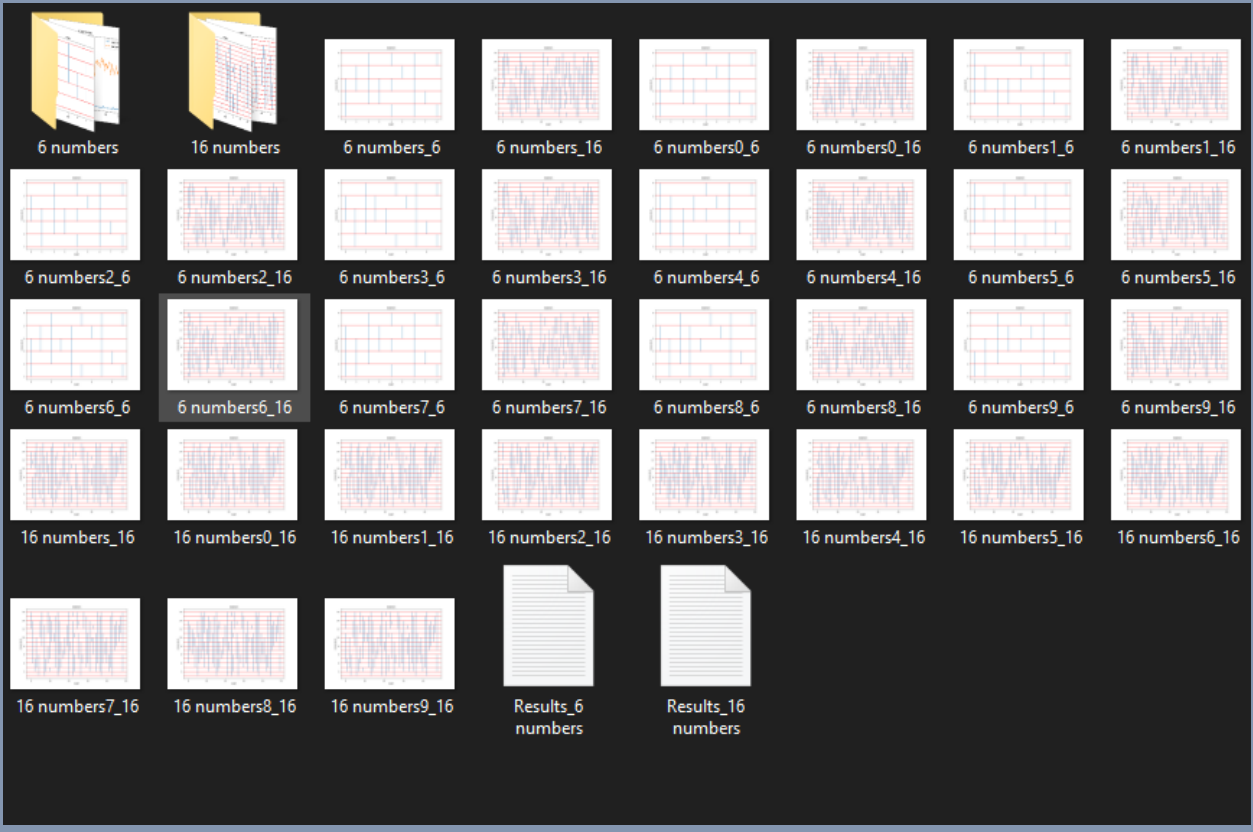

network
comparators
depth
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8


network
comparators
depth
1
2
3
4
5
6
0
1
2
3
4
5
6
7
8


network
comparators
depth
1
2
3
4
5
6
0
2
4
6
8



All output files that are not in a folder are for the top 10 performers in each problem set:



Output format and demonstration of the depth dipping to the optimal :

```
Best:depth 1: (10,14)(7,13)(9,15)
depth 2: (12,13)(6,9)(10,14)(8,16)(4,11)(3,5)
depth 3: (3,14)(11,16)(4,8)
depth 4: (2,3)(7,13)(9,14)
depth 5: (7,16)(3,15)
depth 6: (1,16)(6,13)(2,5)(8,10)(12,14)(3,9)
depth 7: (5,7)(6,10)(8,13)
depth 8: (7,12)(9,14)(1,13)(15,16)
depth 9: (5,14)(2,6)(10,11)(12,16)(3,8)
depth 10: (9,10)(5,12)(2,8)
depth 11: (1,5)(7,12)
depth 12: (5,16)
depth 13: (10,16)(3,6)(5,15)
depth 14: (3,5)(8,10)
depth 15: (7,10)(3,12)
depth 16: (3,7)(2,5)(6,13)(10,14)
depth 17: (6,8)(2,10)(4,11)(1,3)
depth 18: (5,6)(1,14)
depth 19: (5,9)(3,4)(12,16)
 ,fittness: 311.1111111111111  Time :  3.3782651000000072  ticks: 3.378023386001587
Best:depth 1: (7,9)(15,16)(6,14)
depth 2: (7,14)(1,8)(4,11)(3,10)(6,15)
depth 3: (1,11)(3,4)(9,16)
depth 4: (11,16)(4,14)(8,13)(5,6)(1,10)
depth 5: (4,5)(3,9)(8,12)
depth 6: (8,9)(2,10)(6,13)(5,7)(4,15)
depth 7: (2,5)(8,11)
depth 8: (6,8)(12,13)(7,10)(5,14)
depth 9: (14,16)(6,15)(10,13)
depth 10: (12,15)(6,14)(7,11)(8,10)(1,5)
depth 11: (5,7)(4,9)(1,3)(12,16)
depth 12: (6,16)(8,13)(4,14)
depth 13: (13,15)(11,16)
depth 14: (6,15)(2,9)(3,12)
depth 15: (12,15)(2,4)(7,13)
depth 16: (12,13)(8,15)(1,5)(2,14)(10,16)
depth 17: (2,15)(8,11)(5,14)
 ,fittness: 288.8888888888889  Time :  6.169659600000003  ticks: 6.169593095779419
Best:depth 1: (10,14)(7,13)(9,15)
depth 2: (12,13)(6,9)(10,14)(8,16)(4,11)(3,5)
depth 3: (6,15)(11,16)(4,8)(2,3)(7,13)(9,14)
depth 4: (7,16)(3,15)
depth 5: (1,16)(6,13)(2,5)(8,10)(12,14)(3,9)
depth 6: (9,14)(4,11)(12,13)(3,6)
depth 7: (1,3)(4,9)(2,16)
depth 8: (1,9)(5,8)(4,7)
depth 9: (7,12)(13,14)(5,11)(1,9)(4,6)
depth 10: (12,16)(3,15)(4,11)(2,14)(9,10)
depth 11: (12,13)(4,16)(8,11)
depth 12: (13,16)(4,10)
depth 13: (1,16)(8,12)(3,14)
depth 14: (3,6)(7,9)(2,12)(4,8)
depth 15: (7,16)(12,15)
depth 16: (2,7)(4,5)(1,13)(12,14)
 ,fittness: 277.77777777777777  Time :  9.049199400000006  ticks: 9.048791646957397
```

Conclusion:

1. We can improve on the algorithm farther by using a basic solution as a base for creating population
2. We can use a different model of the GA as we have used a type of “species” GA
3. The fitness function of the depth might need to be given less control over the fitness of the networks maybe a 20-30% affect on it would be better
4. The regret that we have in this assignment is that we put ourselves under a lot of pressure , we are amazed by the time management skills that we acquired in this endeavor .

Instruction on usage :

1. just like all old assignments ,all the outputs are generated by a script that can be chosen in the main menu
2. you are given two options either manual or automatic ,
3. press the correct number to choose the aforementioned mode

Lastly , all the output files are generated automatically in the output folder :

Includes: results_X_numbers : this folder has the output of the top 10 solutions for k=16,6

Thank you for reading , we hope that you enjoyed