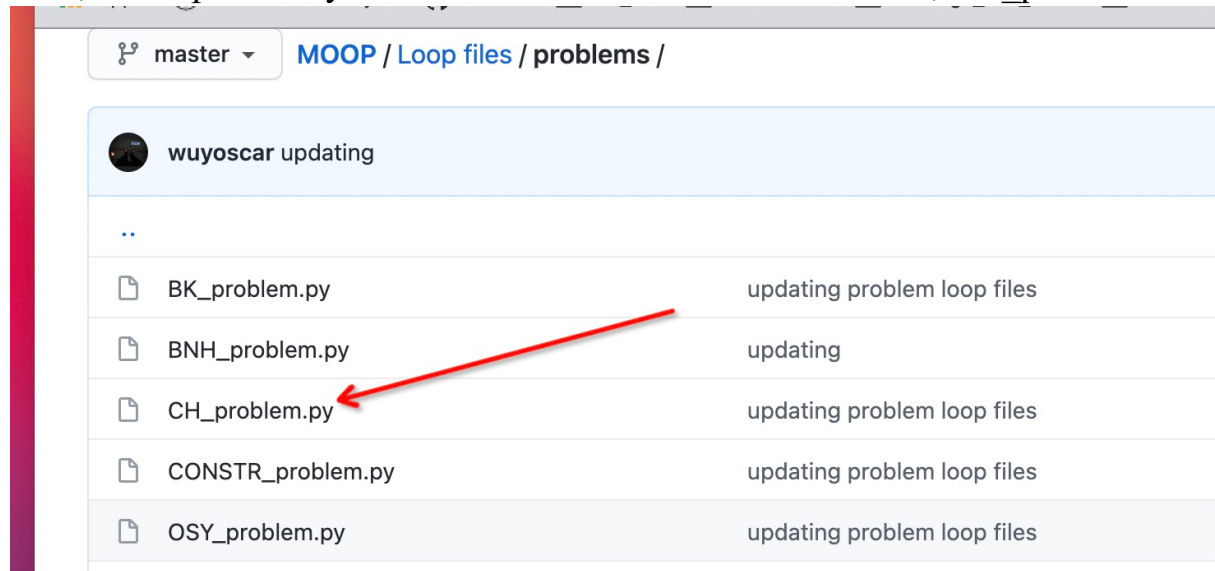


## Week 3 Report

The remote repository has been updated. At present, there is a 'Loop files' Folder that includes: a) problem loop files b) algorithms loop files.

### For problems loop files

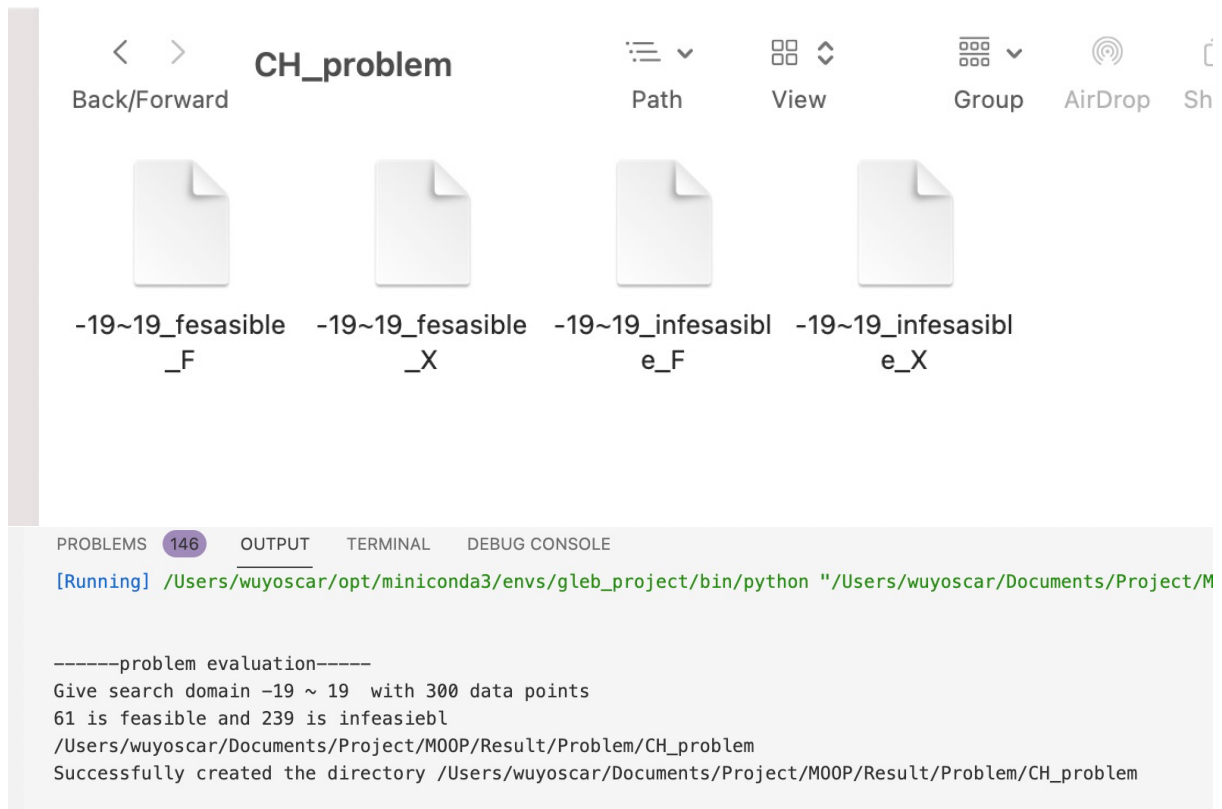
First, chose problem you want for random search. In this case, CH\_problem



Secondly, for all problems files, you need to define two parameters: Bound and number of data points. In this case, upper bound and lower bound random generate by numpy. There are many ways to define your bound. For number of data points, select number. In this case, it is 300.

```
62
63 #get problem
64
65 p = 'CH_problem'
66
67 problem = changkong_and_haimes_problem()
68
69 #!define bound here
70 ub = np.random.randint(20) #upper bound
71 lb = - np.random.randint(20) # Lower bound
72
73 X = random_pick_X(n_var = problem.n_var, bound = [lb,ub], datasize=300) #! define data size here
74 print('\n')
75
76
77 problem_result = problem.evaluate(X)
```

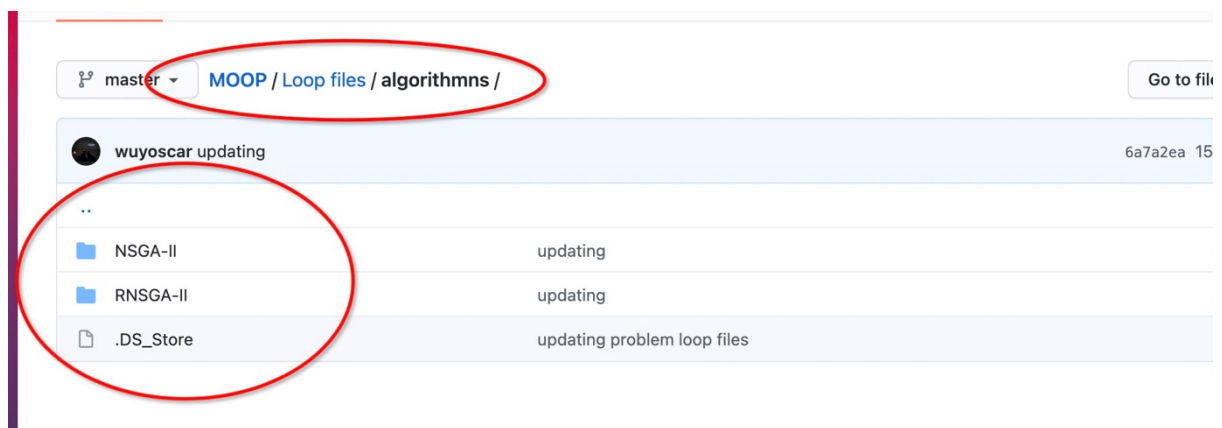
Finally, each time running this file, you will get result shown as below (they are automatic stored in the result folder):



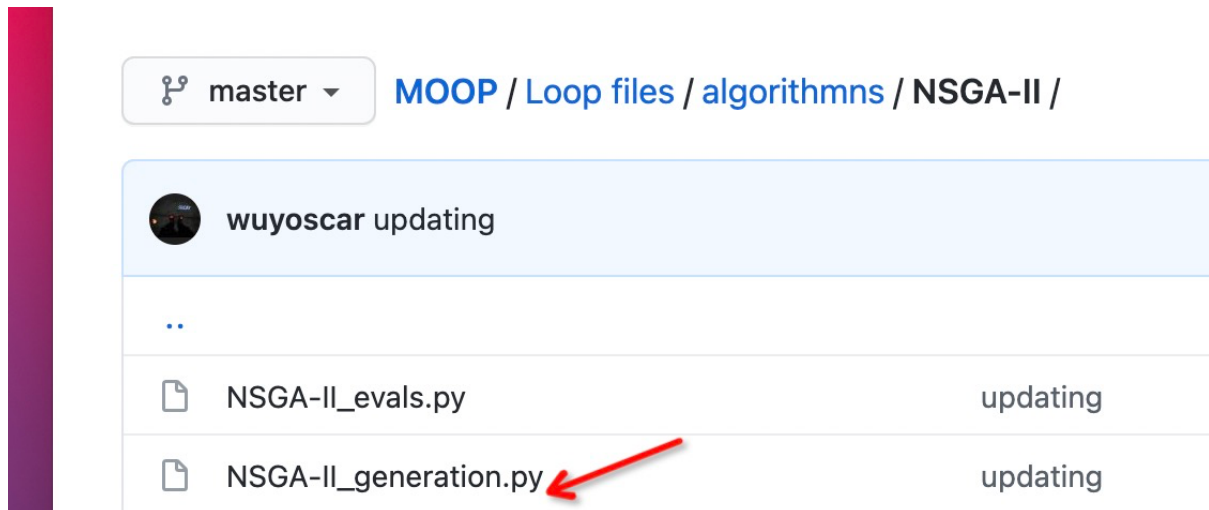
### Algorithms loop files:

Each algorithm loop file is trying to fix the parameters in terms of different problems:

Find files here:



For example: I would like to compare how different number of generation times affect the algorithm performance in terms of NSGA-II:



By select NSGA-II\_generation.py file.

Note: Each algorithm file has multiple loop function inside: you need define some parameter lists in prior:

**Problem list** is available problems can be used here. They are all multi\_problems

Selecting problem you would like to have and pass they into **select\_problem\_list**


```
12
13 #for path in sys.path:
14 #    print(path)
15
16 problem_list = ['BNH', 'OSY', 'TNK', 'Truss2D', 'Welded_Beam', 'zdt1', 'ZDT2', 'ZDT3', 'ZDT4', 'ZDT5', 'ZDT6']
17
18 #!select problem here
19 select_problem_list = ['BNH', 'TNK']
20
21
```

And then pass the generation number list you would like to compare below:

```

22 problem_parameter_dict = {}
23
24 #! set generation times list here
25 #####!!!!!!!!!!!!!!
26 gen_list = [10,20,80,100,200,300,500,800,900,1000]
27 for problem_name in select_problem_list:
28     res_dict = {}
29
30     for n_gen in gen_list:
31         print('\n\n')
32         print('='*60)
33
34         p = get_problem(problem_name)
35         print('problem is:\n',p)
36
37         #algorithm parameters
38         algorithm = NSGA2(
39             n_offspring = 20,

```



The file working flow:

1. Selecting problem
2. For selected problem, loop different generation number in NSGA-II (in this case), it's NSGA-II parameter, because example is NSGA-II\_generation.py
3. For each generation number, calculate gradient distance between pf and object vector generated by algorithm. The smaller the distance is, the better result is. Alternatively, we can use other performance indicators such as IGD, hypervolume etc.

Result show below:

```

=====
problem is:
# name: TNK
# n_var: 2
# n_obj: 2
# n_constr: 2

Time elapsed for solving problem: 6.902482986450195 seconds

Generation times 1000
GD 0.00493776712269816
Creation of the directory /Users/wuyoscar/Documents/Project/M00P/Result/Algorithmn_Result
NAGA-II failed
/Users/wuyoscar/Documents/Project/M00P/Result/Algorithmn_Result/NAGA-II/
NSGA-II_TNK_generation_1000_X

!!!!!!Find the best parameter here !!!!!

{'300': 0.0035757368245021947, '200': 0.004046918780755697, '100': 0.004540354863813304,
'500': 0.0046481690665592665, '1000': 0.00493776712269816, '80': 0.004947189746196255,
'900': 0.004969422263055541, '800': 0.005024694312854494, '20': 0.012486891434664225,
'10': 0.020831072333003443}

```

In this case, we tested NSGA-II with TNK problem. If the generation number is 300, the GD is 0.003575, which is best. We looped generation number list 300, 200, 100, 500, 1000, 80, 900, 800, 20, 10.

Further, result will be stored under result folder like below:

```

/Users/wuyoscar/Documents/Project/M00P/Result/Algorithmn_Result/NAGA-II/
NSGA-II_TNK_generation_1000_X

```

Plus, we can loop problems and parameter (i.e generation) at the same time. Result will be like this. In this case, two problems with generation loop list.

```

2 -----
2 {'BNH': {'800': 0.33268769693576317, '900': 0.3725324445205023, '80': 0.
2 37551000444601235, '20': 0.37742743354693753, '300': 0.3820266547147527, '10': 0.
2 4066756614383871, '500': 0.4195672339687046, '100': 0.4230938528458062, '200': 0.
2 43346362703050834, '1000': 0.43380512597119464}, 'TNK': {'300': 0.0035757368245021947,
2 '200': 0.004046918780755697, '100': 0.004540354863813304, '500': 0.0046481690665592665,
'1000': 0.00493776712269816, '80': 0.004947189746196255, '900': 0.004969422263055541,
'800': 0.005024694312854494, '20': 0.012486891434664225, '10': 0.020831072333003443}}

```