In [1]:
```
import sys

sys.path.insert(1, '../../src')

from ce.tsp_optimized import create_tsp as create_tsp_np
from ce.tsp import create_tsp
from ce.algorithms.greedy_regret_heuristics import *
from ce.algorithms.greedy_heuristics import *
from ce.utils.plot import quality_plots
from ce.utils.experiments import experiment, run_all_experiments
import random
from ce.algorithms.greedy_regret_heuristics.greedy_cycle_with_regret import greedy_cy
from ce.algorithms.greedy_regret_heuristics.greedy_cycle_with_weighted_regret import

random.seed(13)
```

Greedy heuristics with regret

Nina Zukowska 148278, Antoni Solarski 148270

In [2]:
```
problem_instance_A_path = '../../data/TSPA.csv'
problem_instance_B_path = '../../data/TSPB.csv'
problem_instance_C_path = '../../data/TSPC.csv'
problem_instance_D_path = '../../data/TSPD.csv'
```

In [3]:
```
tspa_np, tspb_np, tspc_np, tspd_np = create_tsp_np(problem_instance_A_path), create_t
tspa, tspb, tspc, tspd = create_tsp(problem_instance_A_path), create_tsp(problem_inst
```

Greedy with regret

Greedily acquires new nodes by the value of 'regret'. Here is the pseudocode:

```
function get_2_regret(edges, tsp, node_index):
    Calculate changes in objective function for node insertion between
diferent nodes

function extend_cycle(cycle, tsp):
    Extend the cycle by adding the node with the maximum regret

function greedy_cycle_with_regret(tsp, start_node):
    Initialize the solution with the start node
    While the solution length is less than desired:
        Extend the solution using the node with max regret
    Return the solution
```

For the sake of this algorithm the previous code was adjusted in order to avoid long running times

In [4]:
```
debug = []
```

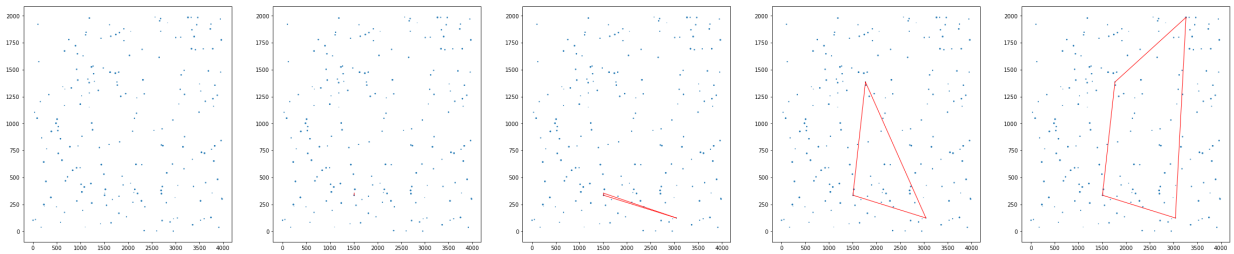
In [5]:
```
%%time
solution = greedy_cycle_with_regret(tspa_np, 0, debug)
```

```
Wall time: 661 ms
```

In [6]:
```
tspa.get_solution_cost(solution)
```

Out[6]:
```
120639
```

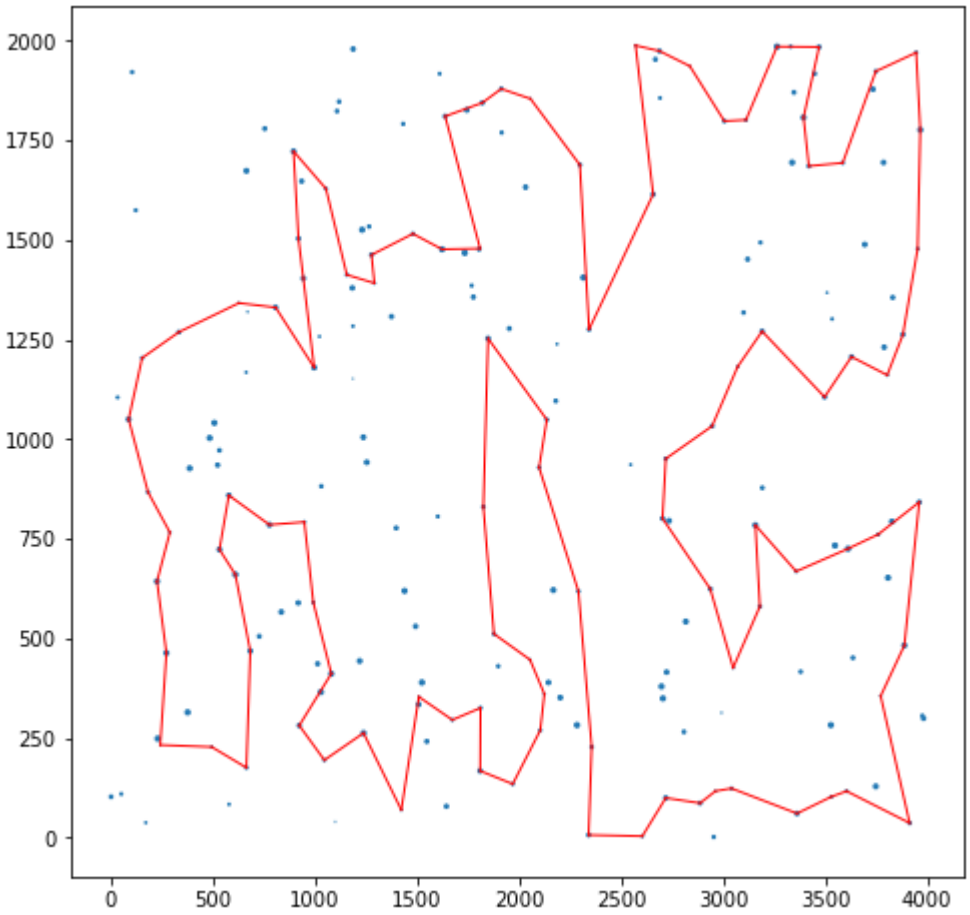
In [7]:
```
tspa_np.plot(debug[0:5])
```



In [8]:
```
%%time
print("Greedy with regret")
regret_results, regret_best = experiment(200, lambda x: greedy_cycle_with_regret(tspa
```

```
Greedy with regret
MIN 109133, AVG 116804.52, MAX 125728
Wall time: 1min 57s
```

In [9]:
```
tspa.plot([regret_best])
```



Greedy with weighted regret

A differnt heuristic for the TSP problem weighting regret with the cost of the node in the final solution. Here is the pseudocode:

```
function get_2_regret_weighted(edges, tsp, node_index):
    Calculate differences for various node insertions, with the cost
of the node subtracted

function extend_cycle(cycle, tsp):
    Extend the cycle by adding the node with the maximum weighted
regret

function greedy_cycle_with_weighted_regret(tsp, start_node):
    Initialize the solution with the start node
    While the solution length is less than desired:
        Extend the solution using the node with max weighted regret
    Return the solution
```

In [10]:
```
debug = []
```

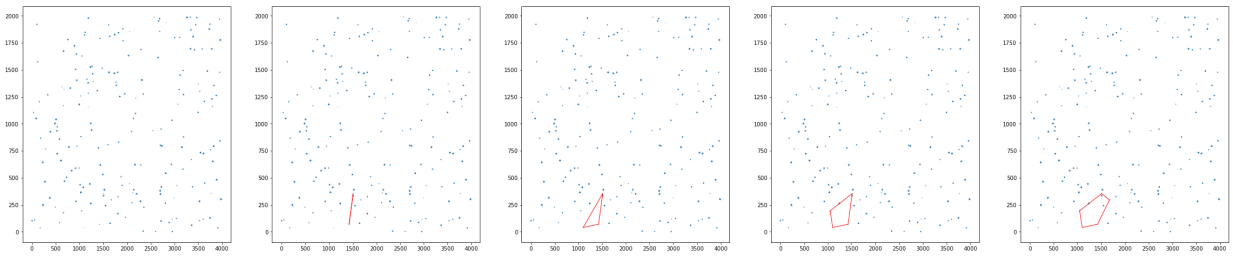
In [11]:
```
%%time
solution = greedy_cycle_with_weighted_regret(tspa_np, 0, debug)
```

```
Wall time: 613 ms
```

In [12]:
```
tspa.get_solution_cost(solution)
```

Out[12]:
```
75628
```

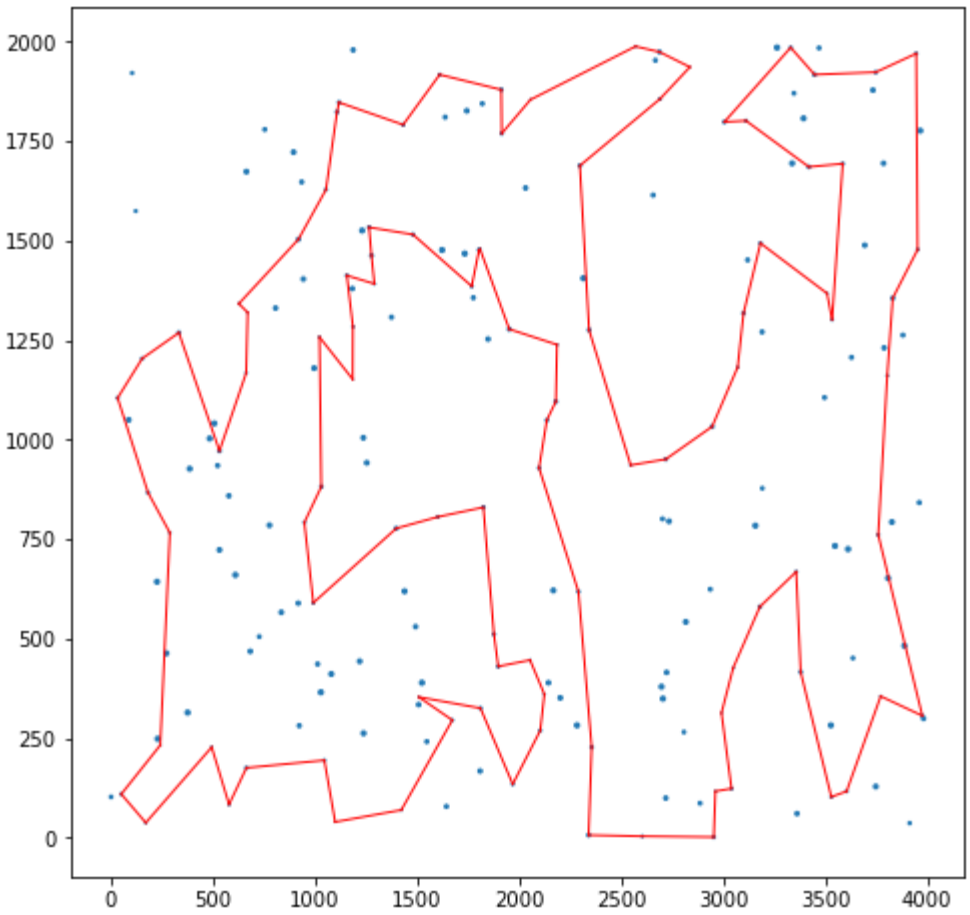
In [13]:
```
tspa.plot(debug[0:5])
```



In [14]:
```
%%time
print("Greedy with weighted regret")
regret_weighted_results, regret_weighted_best = experiment(200, lambda x: greedy_cyc
```

```
Greedy with weighted regret
MIN 74563, AVG 76341.665, MAX 78976
Wall time: 2min 10s
```

In [15]:
```
tspa.plot([regret_weighted_best])
```

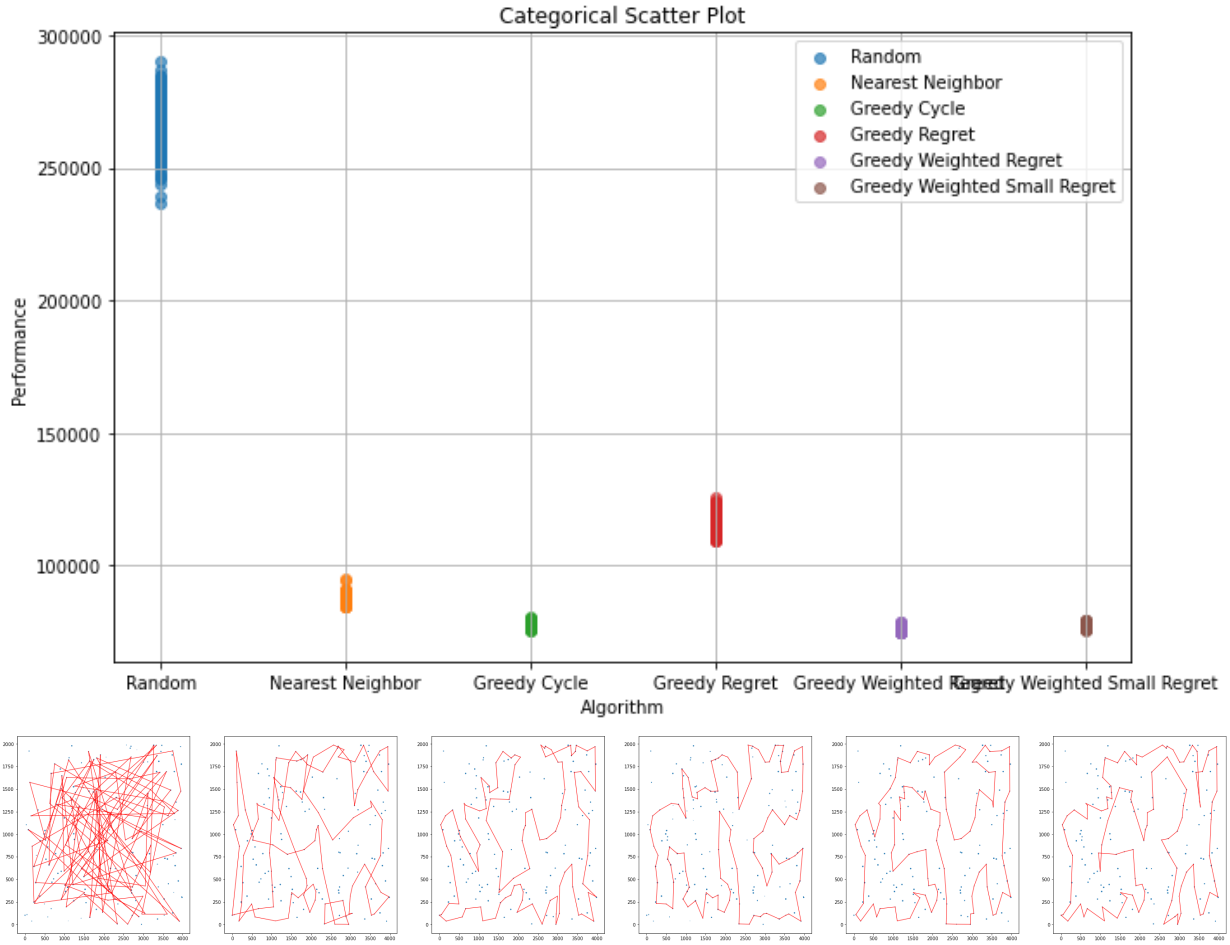


Instance A

In [16]:
```
best_solutions, results_list = run_all_experiments(200, [lambda x: random_solution(ts
                            lambda x: nearest_neighbor(tspa, x),
                            lambda x: greedy_cycle(tspa, x),
                            lambda x: greedy_cycle_with_regret(tspa_np, x),
                            lambda x: greedy_cycle_with_weighted_regret(tspa_np, x),
                            lambda x: greedy_cycle_with_weighted_regret(tspa_np, x, weig
                            ],
                        lambda x: tspa.get_solution_cost(x),
                        ["Random",
                         "Nearest Neighbor",
                         "Greedy Cycle",
                         "Greedy Regret",
                         "Greedy Weighted Regret",
                         "Greedy Weighted Small Regret"])
tspa.plot(best_solutions)
```

```
Running Random
MIN 236587, AVG 264935.945, MAX 290340
Running Nearest Neighbor
MIN 84471, AVG 87679.135, MAX 95013
Running Greedy Cycle
MIN 75666, AVG 77076.88, MAX 80321
Running Greedy Regret
MIN 109133, AVG 116804.52, MAX 125728
Running Greedy Weighted Regret
MIN 74563, AVG 76341.665, MAX 78976
```

```
Running Greedy Weighted Small Regret
```

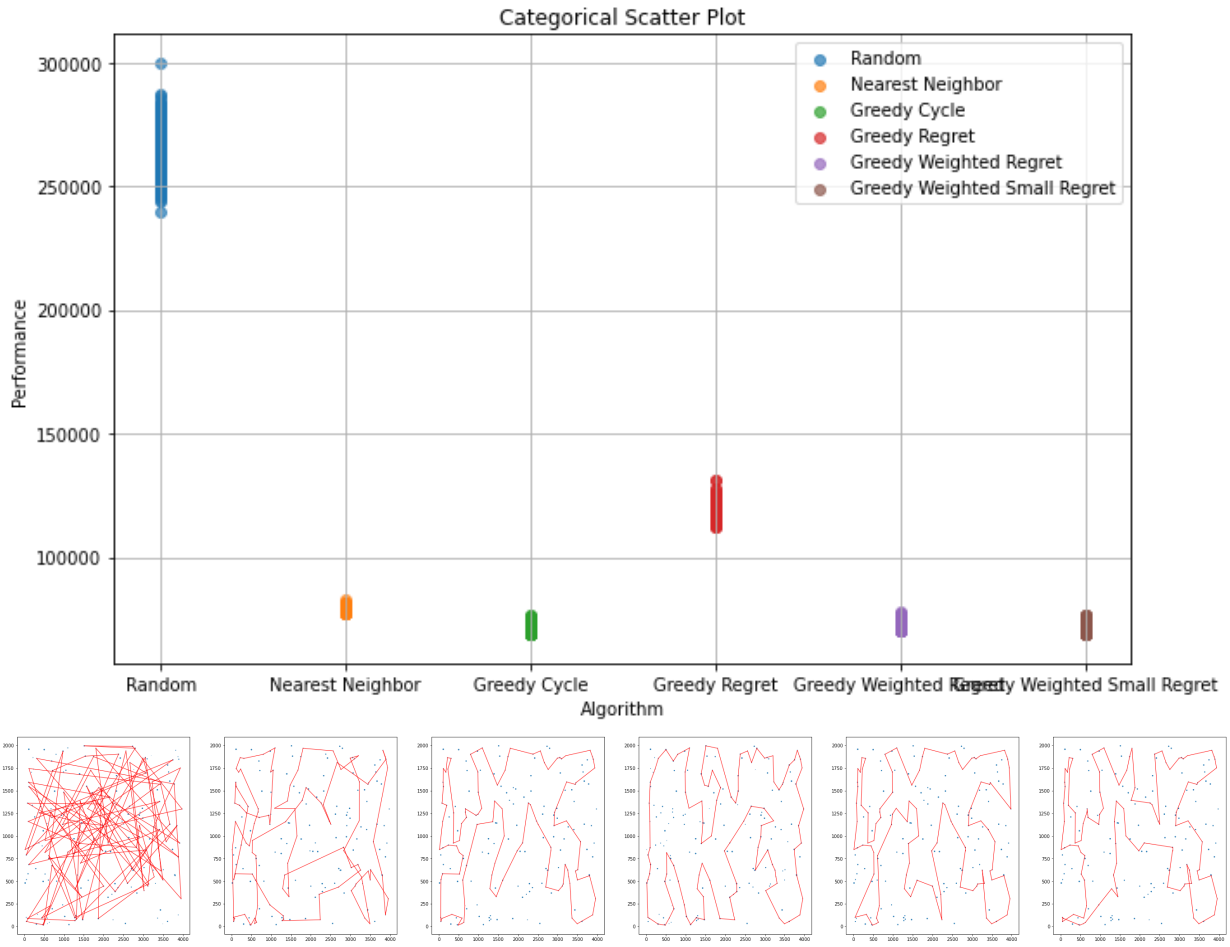


Instance B

In [17]:
```
best_solutions, results_list = run_all_experiments(200, [lambda x: random_solution(ts
                           lambda x: nearest_neighbor(tspb, x),
                           lambda x: greedy_cycle(tspb, x),
                           lambda x: greedy_cycle_with_regret(tspb_np, x),
                           lambda x: greedy_cycle_with_weighted_regret(tspb_np, x),
                           lambda x: greedy_cycle_with_weighted_regret(tspb_np, x, weig
                           ],
                       lambda x: tspb.get_solution_cost(x),
                       ["Random",
                        "Nearest Neighbor",
                        "Greedy Cycle",
                        "Greedy Regret",
                        "Greedy Weighted Regret",
                        "Greedy Weighted Small Regret"])
tspb.plot(best_solutions)
```

```
Running Random
MIN 239845, AVG 265702.055, MAX 299886
Running Nearest Neighbor
MIN 77448, AVG 79282.58, MAX 82631
Running Greedy Cycle
MIN 68743, AVG 70727.43, MAX 76324
Running Greedy Regret
MIN 112088, AVG 119030.635, MAX 131087
```

```
Running Greedy Weighted Regret
MIN 70153, AVG 71801.35, MAX 77676
Running Greedy Weighted Small Regret
```

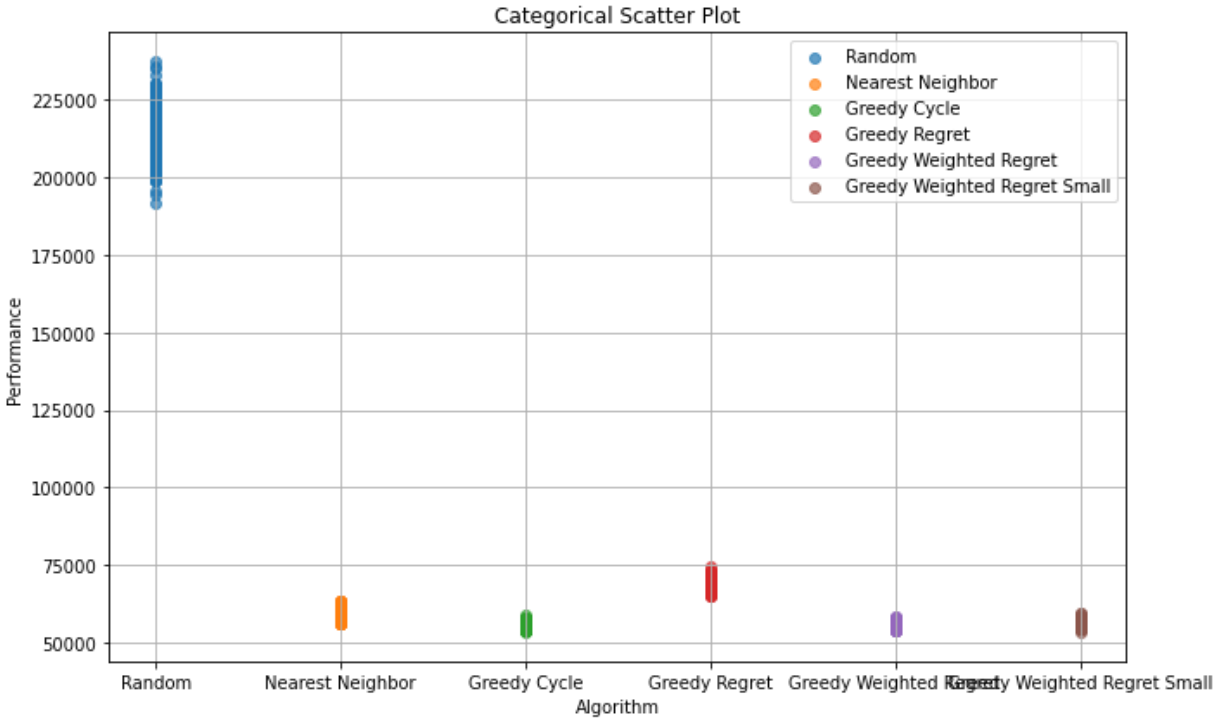


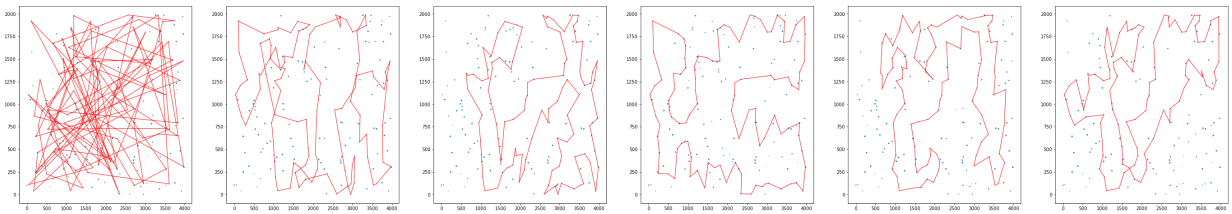
Instance C

In [18]:
```
best_solutions, results_list = run_all_experiments(200, [lambda x: random_solution(ts
                           lambda x: nearest_neighbor(tspc, x),
                           lambda x: greedy_cycle(tspc, x),
                           lambda x: greedy_cycle_with_regret(tspc_np, x),
                           lambda x: greedy_cycle_with_weighted_regret(tspc_np, x),
                           lambda x: greedy_cycle_with_weighted_regret(tspc_np, x, weig
                           ],
                       lambda x: tspc.get_solution_cost(x),
                       ["Random",
                        "Nearest Neighbor",
                        "Greedy Cycle",
                        "Greedy Regret",
                        "Greedy Weighted Regret",
                        "Greedy Weighted Regret Small"])
tspc.plot(best_solutions)
```

```
Running Random
MIN 191455, AVG 214795.975, MAX 237507
Running Nearest Neighbor
MIN 56304, AVG 58872.68, MAX 63697
Running Greedy Cycle
MIN 53226, AVG 55839.8, MAX 58876
```

```
Running Greedy Regret
MIN 65095, AVG 69302.655, MAX 74385
Running Greedy Weighted Regret
MIN 54126, AVG 55946.205, MAX 58288
Running Greedy Weighted Regret Small
```



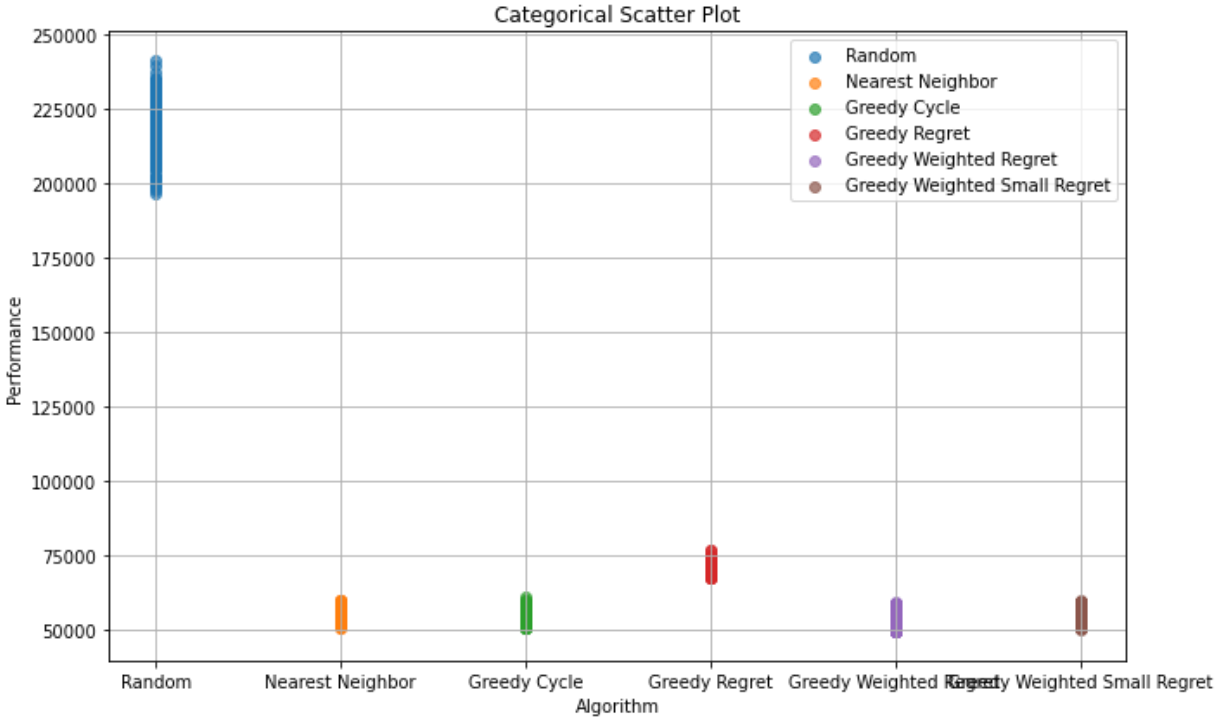


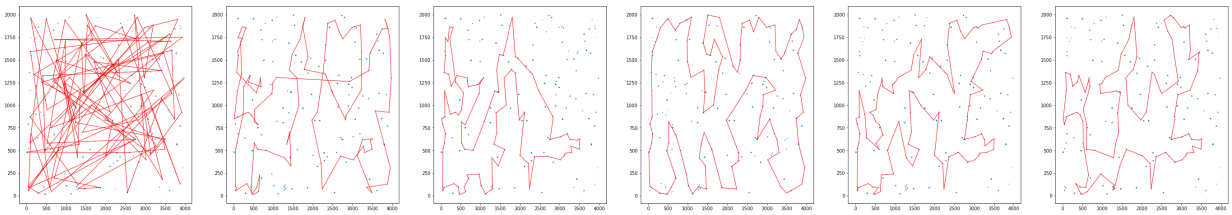
Instance D

In [19]:
```
best_solutions, results_list = run_all_experiments(200, [lambda x: random_solution(ts
                            lambda x: nearest_neighbor(tspd, x),
                            lambda x: greedy_cycle(tspd, x),
                            lambda x: greedy_cycle_with_regret(tspd_np, x),
                            lambda x: greedy_cycle_with_weighted_regret(tspd_np, x),
                            lambda x: greedy_cycle_with_weighted_regret(tspd_np, x, weig
                            ],
                        lambda x: tspd.get_solution_cost(x),
                        ["Random",
                         "Nearest Neighbor",
                         "Greedy Cycle",
                         "Greedy Regret",
                         "Greedy Weighted Regret",
                         "Greedy Weighted Small Regret"])
tspd.plot(best_solutions)
```

```
Running Random
MIN 196786, AVG 218867.88, MAX 241394
Running Nearest Neighbor
MIN 50335, AVG 54290.68, MAX 59846
```

```
Running Greedy Cycle
MIN 50409, AVG 54832.245, MAX 60964
Running Greedy Regret
MIN 67174, AVG 70583.535, MAX 76747
Running Greedy Weighted Regret
MIN 49165, AVG 53706.555, MAX 59416
Running Greedy Weighted Small Regret
```





Conclusions

```
Each instance shows a similar pattern with different tour lengths for
the various methods.
The Nearest Neighbor and Greedy Cycle methods generally perform better
than the random method.
The Greedy Cycle Regret and Weighted Regret methods may have mixed
results, with the regret method sometimes producing longer tours.
Instance-specific characteristics and the specific heuristics used can
affect tour lengths.
```

In summary, the results indicate that different TSP instances yield varying tour lengths based on the chosen construction and improvement heuristics. The choice of heuristic can have a significant impact on the quality of the solutions. Further analysis and experimentation may be needed to determine the most effective heuristic for specific TSP instances.

see code: https://github.com/Antsol1000/ce

In [ ]: