# 09-HEA

January 26, 2025

Made by: Andrei Kulchyk (155489) and Fiodar Piatrovich (155174)

Github

## 1 Description of a problem

We are given three columns of integers with a row for each node. The first two columns contain x and y coordinates of the node positions in a plane. The third column contains node costs. The goal is to select exactly 50% of the nodes (if the number of nodes is odd we round the number of nodes to be selected up) and form a Hamiltonian cycle (closed path) through this set of nodes such that the sum of the total length of the path plus the total cost of the selected nodes is minimized.

The distances between nodes are calculated as Euclidean distances rounded mathematically to integer values. The distance matrix should be calculated just after reading an instance and then only the distance matrix (no nodes coordinates) should be accessed by optimization methods to allow instances defined only by distance matrices.

```
/Users/theodore/Repos/put-evolutionary-computations/.venv/lib/python3.13/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

## 2 Hybrid Evolutionary Algorithm

### 2.1 Psuedo code for HEA

```
Step 1: Generate intial solution
    - Generate random solutions of population_size
    - Apply EdgeSteepestLocalSearch to generated solutions
    - Repeat proceedure, if resulting population is not distinct.

WHILE time_budget not exceeded:

    Step 2: Select parents
        - Randomly select two different parents (parent1, parent2) uniformly from the populatio

    Step 3: Apply recombination
        - IF random.random() < use_operator_1_prob:
```

```
            child_solution = recombination_operator_1(parent1, parent2, all_nodes)
        - ELSE:
            child_solution = recombination_operator_2(parent1, parent2, ds, dm)

    Step 4: Optional local search
        - IF with_local_search_after_recombination is True:
            child_solution = EdgeSteepestLocalSearch(ds, dm, child_solution)

    Step 5: Evaluate child solution
        - Calculate child_cost = function_cost(ds.loc[child_solution])

    Step 6: Update population
        - IF child_cost < max(population_costs) AND child_solution is unique in population:
            Replace the worst solution in the population with child_solution.

RETURN the best solution (minimum cost in population) and num_iterations.
```

### 2.2 Psuedo code for Opearator 1

```
Step 1: Combine common edges
    - Randomly choose either parent1 or parent2 as the starting point.
    - Create child_solution by adding nodes from common_edges in the order they appear in the

Step 2: Fill remaining nodes
    - Identify remaining_nodes = all_nodes - nodes in child_solution.
    - Shuffle remaining_nodes randomly.
    - Add nodes from remaining_nodes to child_solution until its size matches parent1.

RETURN child_solution.
```

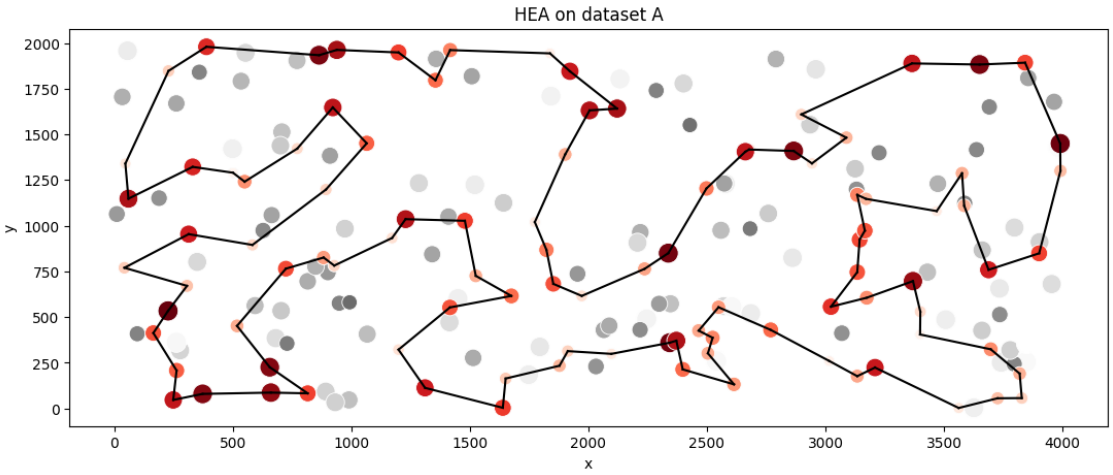
### 2.3 Psuedo code for Opearator 2

```
Step 1: Combine common edges
    - Randomly choose either parent1 or parent2 as the starting point.
    - Create child_solution by adding nodes from common_edges in the order they appear in the

Step 2: Repair the child solution
    - Use init_greedy_2regret_weighted_cycle to repair child_solution:
        - Start with child_solution as the initial solution.
        - Apply the greedy weighted Heurisitc to ensure it forms a valid Hamiltonian cycle.

RETURN repaired child_solution.
```

### 2.4 Results on DataSet A

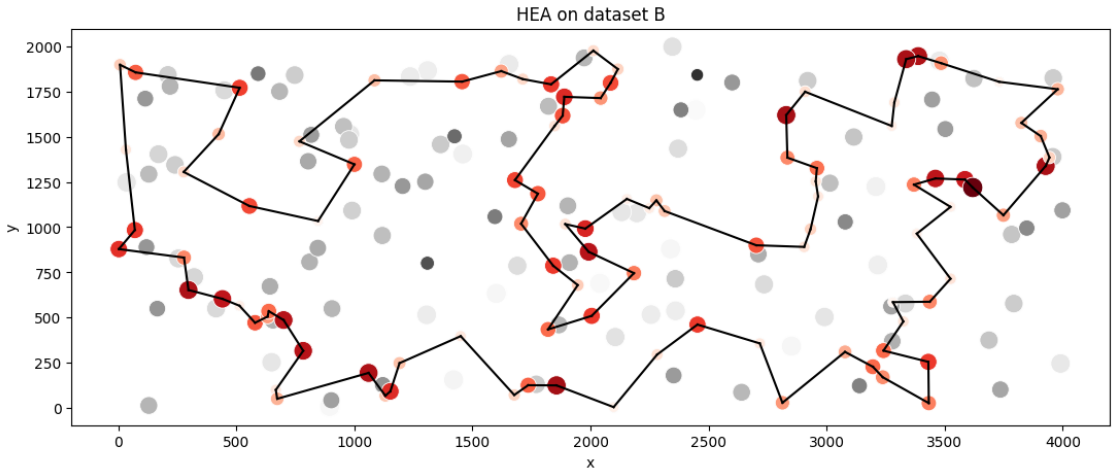
```
Best solution: [154, 180, 53, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25,
16, 171, 175, 113, 31, 78, 145, 179, 92, 129, 57, 55, 52, 178, 106, 185, 165,
40, 196, 81, 90, 27, 164, 7, 21, 144, 14, 49, 102, 62, 9, 148, 124, 94, 63, 79,
80, 176, 137, 23, 186, 89, 183, 143, 0, 117, 93, 140, 108, 18, 22, 146, 159,
```

193, 41, 139, 68, 46, 115, 42, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84,
184, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135]
Objective function statistics:
minimum_a.cost = 69213
mean_a = 69543.65
maximum_a.cost= 69796
Mean Number of iterations: 2705.15




## 2.5 Results on DataSet B

Best solution: [95, 130, 99, 22, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140,
183, 152, 170, 34, 55, 18, 62, 124, 106, 143, 35, 109, 0, 29, 160, 33, 144, 111,
82, 21, 8, 104, 138, 11, 139, 168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6,
147, 90, 51, 121, 131, 135, 122, 133, 107, 40, 63, 38, 27, 1, 156, 198, 117,
193, 31, 54, 73, 136, 190, 80, 45, 175, 78, 5, 177, 36, 61, 91, 141, 77, 81,
153, 187, 163, 89, 127, 137, 114, 103, 113, 176, 194, 166, 86, 185]
Objective function statistics:
minimum_b.cost = 43518
mean_b = 43715.0
maximum_b.cost= 44233
Mean Number of iterations: 2332.8




## 3 Summary

| | Dataset A | | | \ |
|---|---|---|---|---|
| | min | mean | max | seconds/instance |
| ILS | 69107.0 | 69326.150 | 69765.0 | 2223.00 |
| HEA | 69213.0 | 69543.650 | 69796.0 | 2223.00 |
| LNS with LS | 69474.0 | 70179.050 | 71022.0 | 2223.00 |
| LNS without LS | 69657.0 | 70494.500 | 71452.0 | 2223.00 |
| MLSM | 70662.0 | 71267.400 | 71693.0 | 2223.00 |
| Greedy weighted cycle | 71057.0 | 72218.320 | 73587.0 | 0.40 |
| Steepest edge LS | 72046.0 | 74033.715 | 78801.0 | 9.54 |

| | | Dataset B | | \ |
|---|---|---|---|---|
| | iterations | min | mean | max |
| ILS | 1106.20 | 43493.0 | 43783.050 | 44312.0 |
| HEA | 2705.15 | 43518.0 | 43715.000 | 44233.0 |
| LNS with LS | 3027.20 | 43568.0 | 44290.550 | 45011.0 |
| LNS without LS | 6705.10 | 43595.0 | 44507.050 | 45558.0 |
| MLSM | 200.00 | 45321.0 | 45751.250 | 4613.0 |
| Greedy weighted cycle | 1.00 | 45453.0 | 46252.105 | 47884.0 |
| Steepest edge LS | 1.00 | 45393.0 | 48264.780 | 50697.0 |

| | seconds/instance | iterations |
|---|---|---|
| ILS | 2218.00 | 1114.80 |
| HEA | 2223.00 | 2332.80 |
| LNS with LS | 2218.00 | 3058.25 |
| LNS without LS | 2218.00 | 6717.85 |
| MLSM | 2218.00 | 200.00 |

| | | |
|---|---|---|
| Greedy weighted cycle | 0.40 | 1.00 |
| Steepest edge LS | 9.02 | 1.00 |

# 4 Conclusion

Using the proposed Operators' configuration (with slight modifications), the HEA algorithm was tested under the same time budget as ILS and MSLS and run 20 times for each dataset. The results showed that HEA outperformed the other algorithms tested in this course, except for ILS. Incorporating ILS-style perturbations to escape unwanted convergence could be a potential improvement to further enhance HEA's performance.