Evolutionary Computation - Assignment 10 Report

- Kajetan Sulwiński 151954
- Mikołaj Marmurowicz 151956

Problem description

Problem

We were to find a cycle that consisted of exactly 50% of the available nodes, where each node had its own cost along with x and y coordinates. The objective function was a sum of node costs and distances (Euclidean) between each traveled node.

Solution implementation

We have added a new method - our custom approach to the TSP problem.

- **Custom approach**
  - **Input:**
    - `costDistanceInfo` : A symmetric matrix of distances and costs between nodes
    - `nodesInCycle` : A list of nodes that are available
    - `popSize` : The size of a population
    - `timeForGen` : The maximum runtime of the algorithm
  - **Output:**
    - An array of new `cycle` node IDs

1. Initialize population:

```
Create empty population array of size popSize
    Generate four solutions using K-Regret Greedy (weight=0.3)
    Generate the rest using random
    Sort population by ascending cost
```

2. Main loop (until timeForGen):

```
iterCount = 0
    While currTime < timeForGen:
        thresh = min(90, 10*exp(iterCount/750) + 40)  # Penalty threshold
        mutThresh = max(1, 15*exp(-iterCount/500) + 1) # Mutation intensity

        Select parentA using Fitness = 1/cost + boost for costs < 69800
```

```
        Select parentB from remaining population

        child = recombinationComplex(parentA, parentB)
        nodesFromParentsCount = size of common components

        For mutThresh iterations:
            Apply Random mutation inter-move or intra-move

        Apply Candidate Local Search (nnSize=5) to child
        Calculate child.costNoPenalty
        Calculate child.cost = costNoPenalty + 4*(nodesFromParents - thresh)

        If child is unique (cost not in population):
            Replace worst solution if child is better

        iterCount += 1

    return best solution
```

3. Function recombinationComplex(parentA, parentB):

```
Find common path components between parents
    Initialize child with common components
    Fill remaining nodes using K-Regret Greedy (k=0.3*remaining_nodes)
    Return child with nodesFromParentsCount = size of common components
```

Presenting the results

Results presented as minimum, average and maximum of objective function

Presented in a table below, each method and each problem instance is shown.

Method	TSPA	TSPB
Custom	69796.95 (69564 - 70102)	44264.11 (43954 - 44469)
Evolutionary - Complex Recomb. - LS	70044.63 (69829 - 70257)	44466.32 (44015 - 44784)
Evolutionary - Complex Recomb. - No LS	71707.53 (71654 - 71807)	47066.53 (46314 - 47530)
Evolutionary - Simple Recomb. - LS	70891.79 (70534 - 71264)	45118.74 (44676 - 45466)
Greedy LS (Edges) on 2-Regret Weighted 200 runs	71509.42 (70571 - 72485)	50033.92 (45855 - 54814)
Iterated LS 20 runs	69256.11 (69095 - 69614)	43634.53 (43448 - 44215)
LNS No LS 20 runs	70097.05 (69336 - 71100)	44849.16 (43961 - 47055)
LNS With LS 20 runs	70020.58 (69373 - 71128)	44481.84 (43845 - 45540)
Multiple start LS 20 runs	71250.74 (70684 - 71957)	45795.84 (45108 - 46295)
Steepest LS (Edges) on 2-Regret Weighted 200 runs	71470.14 (70510 - 72614)	49895.7 (45867 - 54814)

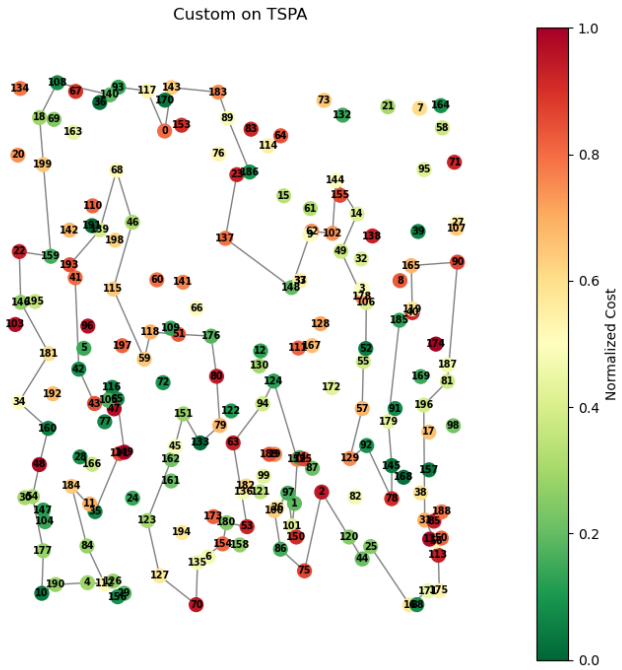
Information regarding running time and iterations of main loop of different methods.

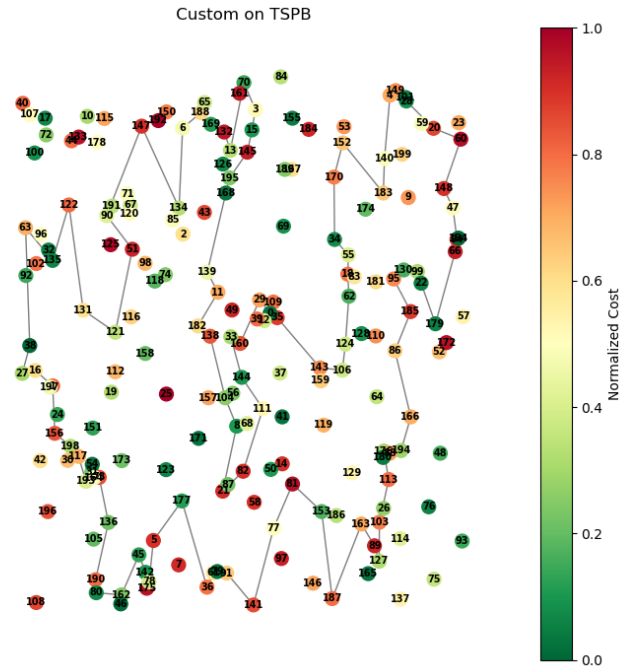
Method	TSPA	TSPB
Custom	805.35 (643 - 1135) runs	1026.25 (984 - 1082) runs
Evolutionary - Complex Recomb. - LS	173.1 (164 - 188) runs	266.7 (256 - 279) runs
Evolutionary - Complex Recomb. - No LS	2397.2 (2284 - 2536) runs	3043.95 (2898 - 3163) runs
Evolutionary - Simple Recomb. - LS	133.7 (124 - 197) runs	204.35 (201 - 207) runs
Iterated LS	2602.95 (2358 - 2823) runs	2611.65 (2416 - 2886) runs
LNS No LS	4271.3 (4105 - 4412) runs	3747.8 (2524 - 4215) runs
LNS With LS	3460.75 (2903 - 3698) runs	2837.95 (1847 - 3562) runs
Multiple start LS	36404.82 (33524.3 - 38601.8) ms	34441.575 (33030.7 - 38215.2) ms

Visualization of the best path for each method

Additionally, a list of node indices is presented.

```
Custom
TSPA
[120, 44, 25, 16, 171, 175, 113, 56, 31, 196, 81, 90, 165, 40, 185, 179, 145, 78, 92, 129, 57
, 55, 52, 106, 178, 49, 14, 144, 102, 62, 9, 148, 137, 23, 186, 89, 183, 143, 0, 117, 93, 140
, 108, 18, 159, 22, 146, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 184, 35, 131, 149, 6
5, 116, 43, 42, 41, 193, 139, 68, 46, 115, 59, 118, 51, 176, 80, 79, 133, 151, 162, 123, 127,
70, 135, 154, 180, 53, 63, 94, 124, 152, 1, 101, 97, 26, 100, 86, 75, 2]
TSPB
[168, 195, 145, 15, 3, 70, 13, 132, 169, 188, 6, 134, 147, 90, 51, 121, 131, 122, 135, 63, 38
, 27, 16, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 162, 45, 142, 175, 78, 5, 177, 36,
61, 91, 141, 77, 81, 153, 187, 163, 89, 127, 103, 113, 176, 194, 166, 86, 185, 95, 130, 99, 2
2, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 14
3, 35, 109, 0, 29, 160, 33, 144, 111, 82, 21, 8, 104, 138, 182, 11, 139]
```





Additional Information

Solution checker

We have checked all of the best solutions via the solution checker provided.

Source code link

The source code is available in a repository here under the Lab10 folder.

Conclusions

While other solutions may have resulted in better quality of solutions, we have decided to explore further the capabilties of evolutionary algorithms and their extensions. Our custom approach demonstrates competitive performance in terms of both solution quality and computational efficiency. The use of adaptive thresholds (thresh and mutThresh) and a focus on incorporating common path components

from parents ensures the generation of high-quality offspring while maintaining diversity in the population. The method uses K-Regret Greedy initialization, which produces a strong starting population. Usage of Candidate Local Search enchances the offsprings, which results in better solutions.

The Custom approach achieves a balance between solution quality and runtime, while also ensuring the stability. Despite the fact that it does not outperform every method in terms of solution quality, it is able to produce relatively good results in a relatively short runtime.

Authors

- Kajetan Sulwiński 151954
- Mikołaj Marmurowicz 151956